

Please type a plus sign (+) inside this box → ☐

09-05-00

PTO/SB/05 (4/98)

Approved for use through 09/30/2000. OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY

PATENT APPLICATION TRANSMITTAL

Attorney Docket No. ETS-TCA

First Inventor or Application Identifier PETER BRITTINGHAM

Title COMPUTER-BASED TEST-ITEM GENERATION AND CLONING

Express Mail Label No. EI469815916US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO:

Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

1. ☒ * Fee Transmittal Form (e.g., PTO/SB/17)
(Submit an original and a duplicate for fee processing)
2. ☒ Specification [Total Pages 79]
(preferred arrangement set forth below)
- Descriptive title of the Invention
 - Cross References to Related Applications
 - Statement Regarding Fed sponsored R & D
 - Reference to Microfiche Appendix
 - Background of the Invention
 - Brief Summary of the Invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claim(s)
 - Abstract of the Disclosure
3. ☒ Drawing(s) (35 U.S.C. 113) [Total Sheets 118]
4. ☒ Oath or Declaration [Total Pages]
- a. ☐ Newly executed (original or copy)
- b. ☐ Copy from a prior application (37 C.F.R. § 1.63(d))
(for continuation/divisional with Box 16 completed)
- i. ☐ DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

5. ☐ Microfiche Computer Program (Appendix)
6. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
- a. ☐ Computer Readable Copy
- b. ☐ Paper Copy (identical to computer copy)
- c. ☐ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. ☐ Assignment Papers (cover sheet & document(s))
8. ☐ 37 C.F.R. § 3.73(b) Statement of Power of Attorney (when there is an assignee)
9. ☐ English Translation Document (if applicable)
10. ☐ Information Disclosure Statement (IDS)/PTO-1449 [Copies of IDS Citations]
11. ☐ Preliminary Amendment
12. ☒ Return Receipt Postcard (MPEP 503)
(Should be specifically itemized)
13. ☐ * Small Entity Statement(s) filed in prior application, Status still proper and desired (PTO/SB/09-12)
14. ☐ Certified Copy of Priority Document(s) (if foreign priority is claimed)
15. ☒ Other: VISUAL BASIC SOURCE CODE APPENDIX (469 SHEETS)
PROLOG SOURCE CODE APPENDIX (95 SHEETS)
COVER SHEETS FOR ALL DOCUMENTS TABLE OF CONTENTS
FOR SPECIFICATION AND SOURCE CODE APPENDICES

NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

16. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP)

of prior application No: _____

Prior application information: Examiner _____

Group / Art Unit: _____

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

☐ Customer Number or Bar Code Label

(Insert Customer No. or Attach bar code label here)

or ☒ Correspondence address below

Name Michael I. Chakansky, Esq.
Sills Cummis Radin Tischman Epstein & Gross PC

Address One Riverfront Plaza
Newark, NJ 07102

City Newark

State NJ

Zip Code 07102

Country USA

Telephone 973-643-5875

Fax 973-643-6500

Name (Print/Type) Michael I. Chakansky

Registration No. (Attorney/Agent)

31,600

Signature

Date

9/1/00

Border Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2000

Patent fees are subject to annual revision.
Small Entity payments must be supported by a small entity statement,
otherwise large entity fees must be paid. See Forms PTO/SB/09-12.
See 37 C.F.R. §§ 1.27 and 1.28.

TOTAL AMOUNT OF PAYMENT (\$ 846

Complete if Known

Application Number	
Filing Date	9/1/00
First Named Inventor	PETER BRITTINGHAM
Examiner Name	
Group / Art Unit	
Attorney Docket No.	ETS-TCA

METHOD OF PAYMENT (check one)

1. ☒ The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number 05-0426

Deposit Account Name EDUCATIONAL TESTING SERVICE

☒ Charge Any Additional Fee Required
Under 37 CFR §§ 1.16 and 1.17

2. ☐ Payment Enclosed:

☐ Check ☐ Money Order ☐ Other

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
101 690	201 345	Utility filing fee	690
106 310	206 155	Design filing fee	
107 480	207 240	Plant filing fee	
108 690	208 345	Reissue filing fee	
114 150	214 75	Provisional filing fee	

SUBTOTAL (1) (\$ 690

2. EXTRA CLAIM FEES

Total Claims	Extra Claims	Fee from below	Fee Paid
20	-20** = 0	9	0
5	-3** = 2	78	156
Multiple Dependent			

**or number previously paid, if greater; For Reissues, see below

Large Entity Small Entity

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description
103 18	203 9	Claims in excess of 20
102 78	202 39	Independent claims in excess of 3
104 260	204 130	Multiple dependent claim, if not paid
109 78	209 39	** Reissue independent claims over original patent
110 18	210 9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) (\$ 846

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
105 130	205 65	Surcharge - late filing fee or oath	
127 50	227 25	Surcharge - late provisional filing fee or cover sheet.	
139 130	139 130	Non-English specification	
147 2,520	147 2,520	For filing a request for reexamination	
112 920*	112 920*	Requesting publication of SIR prior to Examiner action	
113 1,840*	113 1,840*	Requesting publication of SIR after Examiner action	
115 110	215 55	Extension for reply within first month	
116 380	216 190	Extension for reply within second month	
117 870	217 435	Extension for reply within third month	
118 1,360	218 680	Extension for reply within fourth month	
128 1,850	228 925	Extension for reply within fifth month	
119 300	219 150	Notice of Appeal	
120 300	220 150	Filing a brief in support of an appeal	
121 260	221 130	Request for oral hearing	
138 1,510	138 1,510	Petition to institute a public use proceeding	
140 110	240 55	Petition to revive - unavoidable	
141 1,210	241 605	Petition to revive - unintentional	
142 1,210	242 605	Utility issue fee (or reissue)	
143 430	243 215	Design issue fee	
144 580	244 290	Plant issue fee	
122 130	122 130	Petitions to the Commissioner	
123 50	123 50	Petitions related to provisional applications	
126 240	126 240	Submission of Information Disclosure Stmt	
581 40	581 40	Recording each patent assignment per property (times number of properties)	
146 690	246 345	Filing a submission after final rejection (37 CFR § 1.129(a))	
149 690	249 345	For each additional invention to be examined (37 CFR § 1.129(b))	
Other fee (specify) _____			
Other fee (specify) _____			
* Reduced by Basic Filing Fee Paid			SUBTOTAL (3) (\$)

SUBMITTED BY

Name (Print/Type)	Michael I. Chakansky	Registration No. (Attorney/Agent)	31,600	Telephone	973-643-5875
Signature		Date	9/1/00		

WARNING:

Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0. the amount of time you are required to complete this form DO NOT SEND FEES OR COMPLETED FORMS TO THE

EI469815916US

pending upon the needs of the individual case. Any comments on an Officer, Patent and Trademark Office, Washington, DC 20231. ommissioner for Patents, Washington, DC 20231.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT EXAMINING OPERATION

ATTN'Y DOCKET NO.: ETS-TCA

APPLICATION OF: PETER BRITTINGHAM, MARY E. MORLEY, MARK K.
SINGLEY, MARK G. ZELMAN, KRISHNA N. JHA,
JAMES H. FIFE, ROBERT L. RARICH, IRVIN R.
KATZ, RANDY E. BENNETT

FOR: COMPUTER-BASED TEST-ITEM GENERATION AND
CLONING

PATENT SPECIFICATION

TABLE OF CONTENTS

BACKGROUND OF THE INVENTION	1
FIELD OF THE INVENTION	1
SUMMARY OF THE INVENTION	3
BRIEF DESCRIPTIONS OF THE DRAWINGS	6
DETAILED SUMMARY OF THE INVENTION	9
THE COMPUTER ENVIRONMENT	9
PROLOG IV	9
CREATING A NEW TEST ITEM	10
NEW FAMILY PROPERTIES DIALOG BOX	11
MULTIPLE CHOICE MODEL	13
CREATING A TEST ITEM MODEL	15
DEFINING VARIABLES BY INDICATING VALUES THE VARIABLES CAN TAKE ON	15
DEFINING VARIABLES DIRECTLY IN VARIABLES WINDOW	16
VARIABILIZING BY USING NAMING CONVENTION AND HIGHLIGHTING	16
VARIABLE NAMING CONVENTION FOR USE IN AUTO- DEFINING VARIABLES	16
IDENTIFYING ELEMENTS OF THE TEST ITEM TO BE VARIABILIZED	17
VARIABILIZING BY USING PRIOR UNFROZEN MODELS OR CHILDREN THEREOF	19
EDITING STRING VARIABLES	20
CREATING AND IMPORTING STRING VALUES	22
Using the "Add" Button	22
Using the "Export Strings" and "Import String" Buttons	23
EDITING INTEGER VARIABLES	24
SPECIFYING THE CONSTRAINTS	25
Operators	26
Variables	26
Functions	26
Constraining "IKey"	26

Exporting and Importing Constraints	27
Constraining the Distractors	27
Testing the Constraints	28
GENERATING TEST ITEM VARIANTS	29
WORKING WITH GENERATED VARIANTS	31
ACCEPTING VARIANTS	31
DEFERRING AND DISCARDING VARIANTS	31
CREATING NEW VARIANT MODELS FROM A GENERATED VARIANT	32
ACCEPTED VARIANTS AND NEW FAMILY MODELS	32
WORKING WITH MODELS AND ACCEPTED VARIANTS	33
EDITING THE PROFILE OF A VARIANT	34
The GRE Difficulty Portion of the Profile of a Variant Window	35
WORKING WITH FAMILY MEMBERS	36
Using a New Active Model to Generate Far Variants	36
Creating Still More Models	37
PRINT OPTIONS	38
GRE QUANTITATIVE COMPARISON ITEMS	39
GMAT DATA SUFFICIENCY ITEMS	40
FURTHER EXAMPLES OF ITEM MODELS	40
PROLOG SIMULTANEOUS CONSTRAINT SOLVER	41
HLP4lib.p4	41
PrlgExpr.l	41
PrlgExpr.y	41
hlp4API.h	42
TCA CONSTRAINT LANGUAGE	43
NOTATIONAL CONVENTION	44
TCA CONSTRAINT LANGUAGE IN DETAIL	44
BASIC ELEMENTS	46
Constants	46
Variables	46

Lists	47
Functions	47
Algebraic Expressions (referred to as: AlgExpr)	51
CONSTRAINT SPECIFICATION	53
Type Constraint Specification	53
Optimizable-Relation Specification	53
Precision Specification	55
Relational Constraints (RelExpr)	55
Ranges	56
Enumerated Range	57
if-then-else Constraint	58
if-then-elseif Constraint	59
Freeze Constraint	60
Primitive succeed and fail constraints	60
Combining Constraints	61
WRITING CONSTRAINTS IN TCA CONSTRAINT LANGUAGE	61
SOME TECHNIQUES TO SOLVE CONSTRAINTS IN TCA	63
Variable Type Specification.	63
Range specification.	64
Enumerated-Range Specification.	64
Efficient Solving	64
Representing lists and tables	65
Bidirectionality of functions and operators	66

Constraints are Solved in Order-independent Fashion	66
Constraints are Solved as a Whole	66
Variable Names are the Links to Bind Various Constraints	67
Use of Sets and Ranges	67
Logical Operators	68
Equality by Assigning Same Variable Name	68
VISUAL BASIC SOURCE CODE APPENDIX	69
PROLOG SOURCE CODE APPENDIX	73
CLAIMS	74
ABSTRACT	79

COMPUTER-BASED ITEM GENERATION

This application claims priority from U.S. Provisional Application Ser. No. 60/152,121, filed September 1, 1999, the disclosure of which is incorporated herein by reference. A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office public patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention relates to computer-based technology in the generation of test items. In particular, to the semi-automated (i.e. test-developer assisted) generation of surface-level (near), medium-level, and deep (far) clones ("variants") of test items.

Applicants' assignee, Education Testing Service administers many different tests, including the following three tests. The Graduate Management Admission Test® (GMAT®) Program is used by

graduate business schools and measures general verbal,
mathematical, and analytical writing skills that an individual
has developed over a long period of time. The Graduate Record
Examinations® (GRE®) Program provides tests that assist graduate
5 schools and departments in graduate admissions activities. Tests
offered include the General Test, which measures developed
verbal, quantitative, and analytical abilities, and the Subject
Tests, which measure achievement in 14 different fields of study.
The SAT® Program consists of the SAT I: Reasoning Test and SAT
10 II: Subject Tests. The SAT I is a three-hour test, primarily
multiple-choice, that measures verbal and mathematical reasoning
abilities. The SAT II: Subject Tests are one-hour, mostly
multiple-choice, tests in specific subjects. These tests measure
knowledge of particular subjects and the ability to apply that
15 knowledge. The SAT® Program tests are used by colleges for
admission or placement purposes.

These and other tests require a large number of test items.
However, creating tests items is an expensive and time consuming
process. Therefore, there is a need for a process and system for
20 creating test items in a relatively cost effective and
expeditious manner.

It is an object of the present invention to provide a process and system for the cost effective and expeditious creation of test items.

It is a further object of the present invention to provide a process and system for the cost effective and expeditious generation of test item variants from existing or newly created test items, wherein said test item variants can be used as test items.

SUMMARY OF THE INVENTION

A computerized method and system for creating test items by generating variants of a test item model, comprising the steps of creating a new test item model by identifying elements of an initial test item or a test item model to be variabilized, variabilizing the elements thereby creating test item variables, indicating values the variables can assume, specifying the constraints that define the relationships among the variables, and generating test item variants utilizing a simultaneous constraint solver.

The initial test item can be a pre-existing test item or test item model, a newly created test item or even a conceptual template in the mind of the test item creator. The generated test item variants are displayed to the test item creator. The

test item creator can store and forward acceptable test item variants for later use as test items. Test item models can be stored for later use in generating new test item variants.

In one preferred embodiment of the present invention, a frozen test item model can be extended to create its child which child model can be modified to generate its own test item variants. Moreover, item classification and tracking functions are provided. Test item creators can type in test items, edit them, import graphics, etc. Items that are created are compatible with test delivery software. Item management features allow the test developer to track the location and status of an item throughout its life cycle. In addition, items may be arranged in a searchable and browsable library in terms of whatever conceptual frameworks are used in the automatic generation and/or analysis of items. The text/graphics of these library items can be directly accessible by the item creation tools, i.e. the user is able to edit the text of a library item to create a new item.

One preferred embodiment of the present invention was written in Visual Basic, as well as the PROLOG IV programming language and provides an environment where the user can create a test item model or a family of test item models. For example, with this embodiment of the present invention, referred to as the

"Test Creation Assistant" or "TCA", the user may want to create a single model for a specific purpose, but could find out that it makes sense to have a family of models that have some sort of related theme and therefor TCA includes the notion of test model families.

Although preferred embodiments of the present invention are described below in detail, it is desired to emphasize that this is for the purpose of illustrating and describing the invention, and should not be considered as necessarily limiting the invention, it being understood that many modifications can be made by those skilled in the art while still practicing the invention claimed herein.

BRIEF DESCRIPTIONS OF THE DRAWINGS

FIGS. 1 - 107 show the computer generated screen displays of one preferred embodiment of the present invention.

FIG. 1 shows the initial Test Creation Assistant window of this preferred embodiment and its associated work areas.

FIG. 2 shows the "File" menu options.

FIG. 3 shows the "New family properties" dialog box which appears by clicking on "New" in the "File" menu shown in FIG. 2.

FIG. 4 shows the "Save new family as" dialog box which appears by clicking on the "OK" button in the "New family properties" dialog box shown in FIG. 3.

FIG. 5 shows the result of the user entering "NEWMC" as the name of a family of test items in FIG. 4 and saving the choice.

FIG. 6 shows the TCA Standard Multiple Choice Model Word template of this preferred embodiment.

FIG. 7 shows the stem after the user has entered an initial test item.

FIGS. 8 and 9 show one way to identify elements of the test item to be variabalized using a preferred naming convention.

FIGS. 10-12 show a method for variabalizing and autodefining preidentified test item elements.

FIG. 13 shows the result of auto-defining the variables.

FIGS. 14 - 18 and 24 - 26 show how string variables may be edited in accordance with a preferred embodiment of the invention.

FIGS. 19 - 20 show how string variables may be exported to a file for later use in accordance with a preferred embodiment of the invention.

FIGS. 21 - 23 and 27 - 29 show how integer variables may be edited in accordance with a preferred embodiment of the invention.

FIGS. 30 - 44 show how the variable constraints may be specified in accordance with a preferred embodiment of the invention.

FIGS. 31 - 51 show how test item variants may be generated in accordance with a preferred embodiment of the invention.

FIGS. 52 - 56 show how the user can work with generated variants in accordance with a preferred embodiment of the invention.

FIGS. 56 - 78 show how the user can work with models and accepted variants in accordance with a preferred embodiment of the invention.

FIGS. 79 - 88 shows one way to print variants and the print outs generated by the system, after the user clicks on "Print

All" in Fig. 79, the print outs showing the variables and constraints, test item model, and test item model variants in accordance with a preferred embodiment of the invention.

FIGS. 89 - 91 show screen displays from Quantitative Comparison items in accordance with a preferred embodiment of the invention.

FIGS. 92 - 93 show screen displays from Data Sufficiency items in accordance with a preferred embodiment of the invention.

FIGS. 94 - 106 show screen displays for various item types in accordance with a preferred embodiment of the invention.

FIG. 107 show an overview of the computer architecture for one preferred embodiment of the present invention.

DETAILED SUMMARY OF THE INVENTION

THE COMPUTER ENVIRONMENT

The computer system of the present invention was designed so
5 that people could use it at home as well as on currently
available desktops at work or notebooks. One preferred
embodiment works with Microsoft® Windows 95, 98 or NT. This
embodiment requires Microsoft® WORD 97, PROLOG IV and a Control
Language Program called TCL 7.6, which is available on the
10 Internet at <http://www.scriptics.com>. See the Source Code
Appendix for further details about this embodiment. The present
invention is not limited to the foregoing operating systems,
programming languages and/or software applications, etc. For
example, an extensible markup language editor could be used in
15 place of Microsoft® WORD.

PROLOG IV

Prolog IV, is a compiled constraint programming language.
The Prolog IV language allows the programmer to process a wide
20 variety of constraints describing relations over real and
rational numbers, integers, booleans and lists. The Prolog IV
constraint solving techniques are based on exact and
approximation methods.

PROLOG IV is distributed by PrologIA, Parc Technologique de
Luminary - Case 919, 13288 Marseille cedex 09, France. Further
information about PROLOG IV can currently be found at
<http://prologianet.univ-mrs.fr>. PROLOG IV is a programming
environment.

CREATING A NEW TEST ITEM

In accordance with the preferred embodiment as exemplified
by the software application disclosed in the SOURCE CODE
APPENDICES, the user upon initializing the software application
is presented with the initial Test Creation Assistant window.
FIG. 1.

The initial window is subdivided into several work areas.
One important area is the Microsoft® Word area, which occupies
most of the left side of the initial window. Also important are
the three tabbed areas: "Family Overview"; "Model Workshop"; and
"Generate Variants" and the two pull down menus: "File" and
"Help". FIG. 1.

The Family Overview windows provide information regarding
Family members and Accepted variants and permits the user to Set
Attributes and Print a Family member. The Model Workshop tab
moves the user to areas for creating variabilized test items.

The Generate Variants tab permits the user to generate one or more test item variants. An item variant is a test item automatically generated from an item model, where the item model is comprised of constraints, stem, and key. In this tab, item variants can be displayed, saved or rejected.

Clicking on "File" menu heading in FIG. 1, opens the pull down menu shown on FIG. 2. The menu options are "New", "Open", "Import Locked Item", "Print Setup" and "Exit".

NEW FAMILY PROPERTIES DIALOG BOX

Clicking on "New" brings up the "New family properties" dialog box. FIG. 3. Using this dialog box the user can select the particular family properties for the new test item. Family properties refers to the major properties associated with a test item. In one preferred embodiment, it refers to the type of "Program" (e.g., GMAT®, GRE®, or SAT®), the "Item type", the "Variant proximity" and whether the test item is to be "generic" or "non-generic".

In the New family properties dialog box, using a drop-down menu, the user can select the "Program": GMAT®, GRE®, or SAT®. GRE has been selected. FIG. 3.

In the New family properties dialog box, the user can also select the "Item Type" or format of the question, in this embodiment: Multiple choice (MC), Quantitative comparison (QC),

or Data sufficiency (DS). All three options are shown in FIG. 89; while Multiple choice appears in FIG. 3.

The user can also select the "Variant proximity": the choices are: Near (variants that are easily recognizable as belonging to the same variant family by test developers and test takers, this selection is shown in FIG. 3); Medium (variants that are not easily recognizable as belonging to the same variant family by test developers and test takers); and Far (variants which are difficult to recognize as belonging to the same variant family). Once selected, the user may strive to ensure that generated variants are of the identified proximity.

Finally, the user has the choice of selecting either "Generic" items or "Non-generic" items. Pure items, i.e., test items in a mathematical setting, are Generic as long as they have no distinguishing surface features, such as an unusual figure or table. Real items, i.e., test items based on a real-life situation, are Generic only if the context is commonly used in the text. For example, common context for GMAT includes buying and selling questions, profit questions, stock questions, interest questions, etc. These would not be Generic items for GRE, since the GRE is not aimed at a business school population. Generic items for GRE would include simple rate-time-distance questions, percent questions, etc.

Clicking on the "OK" button in FIG. 3 brings up the "Save new family as" dialog box shown in FIG. 4. The user then enters the name of a family of test items, for example "NEWMC" and clicks on the "Save" action button, the file is saved as a "Model Doc Files (*\$R.doc) and the result is shown in FIG. 5.

At this point the Program (GRE), Family (NEWMC\$R.doc), Attributes (Single multiple choice or SMC; Non generic and Near), and the Active Model (NEWMC\$R.doc) are displayed in status bars at the bottom of the ETS Test Creation Assistant window. This information is displayed to the user across all three tabs: Family Overview (FIG. 5); Model Workshop (FIG. 13); and Generate Variables (FIG. 50).

MULTIPLE CHOICE MODEL.

In the left part of the window in FIG. 5 appears the Microsoft Word document window with titles: "TCA Standard Multiple Choice Model", "reserved for variants", "stem", and "key". (Also present but not shown are the distractor titles and scratch pad, which can be seen in FIG. 6.) The first title will depend on the item type that the user chooses in the "New family properties" dialog box, see for example FIG. 3. The TCA Standard Multiple Choice Model Word template as printed out is shown in FIG. 6. When the user chooses Quantitative comparison, see FIG. 89, the result shown in FIG. 90 is the TCA Quantitative

Comparison Model (see also FIG. 91), if Data Sufficiency is chosen the result shown in FIG. 103 is the TCA Data Sufficiency Model (see also FIG. 105).

In the right part of the window in FIG. 5 the "Family Overview" tab is highlighted. In "Family members" appears an icon with a "Sun" and the name of the variant family, chosen in FIG. 4, "NEWMC", and an extension "\$R.doc". The "R" identifies the test item model as the root model, while the ".doc" identifies the file as a WORD document. The "Sun" icon indicates that the model is active, that is, an item model that has of yet not produce accepted variants and, therefore, is not blocked for future changes.

At the bottom of the "Family members" are two active buttons: "Extend" and "Remove". These buttons enable the user to extend or remove a variant family, respectively.

Creating an item begins with making entries in the "stem" section of the TCA Standard Multiple Choice Model. This is preferably done in the "Model Workshop" environment, but may be started in the "Family Overview" environment as shown in FIG. 7. As shown in FIG. 7, the user entered the following initial test item.

If John has 5 apples and Mary has 6 apples, how many apples do they have together?

CREATING A TEST ITEM MODEL

The present invention provides for the automatic generation
5 of test item variants. To do this the user builds a test item
model. Both the John and Mary initial test item above and an
existing model can form the basis for building a new test item
model. A test item model, whatever its size, consists of
"variables" and "constraints". A constraint is a statement
10 specifying the relationships among the variables. A variable, in
turn, indicates an item element that can take on more than one
value. Variables are defined by the user in terms of type
(integer, real, string, etc.) as well as the value range those
variables can take on. Therefore, to build a new model the user
15 needs to introduce some variables and define a set of constraints
for those variables-- the variabilizing process.

DEFINING VARIABLES BY INDICATING VALUES THE VARIABLES CAN TAKE ON

In accordance with one preferred embodiment of the present
20 invention, there are three ways of starting the variablizing
process: direct entry of information related to the term to be
variabilized, such as, name of variable, type, etc.; by pre-
defining variables in the stem using a naming convention so that
the system will automatically include the variable and its

associated information in the "Variables" window; or by starting with existing test item models or their child models.

DEFINING VARIABLES DIRECTLY IN VARIABLES WINDOW

5 The first method is to go over to the Model Workshop tab, and under the "Variables window" in the Model Workshop click on "Add", a "Create or Change Variable" dialog box will open, similar to that shown in FIG. 22, except that the "Variable Name" box is empty and the user must type the name in himself/herself instead of having the system do it automatically as in the second method.

VARIABILIZING BY USING NAMING CONVENTION AND HIGHLIGHTING

15 The second method is to rename those elements of the initial test item when in the stem in accordance with the naming convention disclosed below, highlight the text typed in the stem section, right click and chose variablize, and the system will allow the user to automatically add all capitalized words which are highlighted.

VARIABLE NAMING CONVENTION FOR USE IN AUTO-DEFINING VARIABLES

When naming variables in the stem for use in auto-defining variables, one preferred embodiment of the present invention uses

the following conventions. Names of the variables are made up of letters and digits; however the first character must be a letter. A string variable begins with an "S"; an integer variable begins with an "I"; a real variable begins with an "R"; a fraction begins with an "F"; and an untyped variable begins with an "U". A "String Variable" is a variable that does text substitutions, it does not involve mathematical operations. The system just searches and replaces one combination of letters, numbers, and/or symbols with another. For example, the text substitutions could be substituting a male name from a list of male names. On the other hand, the text substitutions could be as complex as substituting a model (with all its variables) from a list of models, etc. "Untyped variables" are any variables representing a list, a boolean, or whose type is not known at test-item design time. These variables need not be defined unless they are referenced in the stem or in the distractors. Related information can be found in the TCA CONSTRAINT LANGUAGE section below.

IDENTIFYING ELEMENTS OF THE TEST ITEM TO BE VARIABILIZED

Using the naming conventions above, elements of the test item to be variabilized can be identified. The elements are identified by typing variable identifiers in place of the original test item element. An example of one such

identification is shown in FIG. 8 and FIG. 9. The user changed John to "SMaleName" and Mary to "SFemaleName". The "S" indicating that the variable is a string variable. The user replaced the numbers 5 and 6 with "INum1" and "INum2", respectively. The "I" indicating that the variable is an integer. The user also replaced "apples" with "SItems", another string variable. So the stem portion has become:

"If SMaleName had INum1 SItems and SFemaleName had INum2 SItems, how many SItems would they have together?"

FIG. 9.

The user also changed the key from "Key" to "IKey" and all the distractors from "Distractor_" to "IDistractor_", because he/she is contemplating that all the solutions will be integers.

FIG. 9.

At this point, variables can be defined by highlighting the terms to be variabilized and then right clicking. FIG. 10. In the menu that appears, highlight and click "Variabilize". The result is shown in FIG. 11.

The "New variable detected" dialog box with words "Auto-define variable ...?" appears. The system will try to auto-define any sequence of letters or digits that begins with a capital letter. The system asks the user whether the first word which is

capitalized "If" should be auto-defined as a variable. The user should click on "No", which causes the system to address the next capitalized word "SMaleNames". FIG. 12. The result of clicking on "Yes" with respect to all words to be variabilized is to automatically classify the chosen terms in accordance with the naming convention. All chosen terms then appear in the "Variables" window as is shown in FIG. 13. Providing the user with additional information, not only do the names of the variables appear the "Variables" window, but also their characteristics. For example, string, integer, etc.

VARIABILIZING BY USING PRIOR UNFROZEN MODELS OR CHILDREN THEREOF

The third method is to chose an existing unfrozen model or child model and edit the existing variables and constraints in the Model Workshop. This is an important advantage of the preferred embodiment of the present invention, as it permits the reuse of prior test item models. If the model is frozen, it can still be used by extending the model and having the system create a "child" model of it; if the model is unfrozen, it can be used or, again, the system can create a "child" model of it. In either case, the "Variables" window, as well as other "constraint" windows, in the Model Workshop are populated with

variablized terms, ready for modifying through the editing process. See, FIG. 55 - FIG. 57, FIG. 71 - FIG. 77, and FIG 81.

EDITING STRING VARIABLES

5 At this point, all the selected variables appear in the "Variables" window. Next the variables must be constrained by assigning values or expressions to them. One way of doing this, is to select a particular variable by making a left mouse button click on the chosen variable, then right clicking, which brings
10 up the menu shown in FIG. 14. The user selects "Edit" so as to begin the process of providing constraints to the possible values or expressions the selected variable can assume. However, as can be readily seen from FIG. 14, the system also permits the user to perform other useful functions at this stage.

15 Selecting "Edit" in FIG. 14 brings up the "Create or Change Variable" dialog box of FIG. 15. In accordance with the naming convention, as implemented via the auto-define variabilizing function of the present invention, and as was indicated in the "Variables" window, the variable "SMaleName" in the "Create or
20 Change Variable" dialog box has been classified as a "String" type variable. SMaleName will be a male name selected from a list. The user may at this point, change the variable type, notwithstanding the naming convention, by highlighting and

clicking on any of the other types listed in the "Type" menu of the "Create or Change Variable" dialog box of FIG. 15.

In this preferred embodiment, "Add to checksum" and "Indexed" boxes are selected by default. Selecting the checksum option helps ensure that a value of a variable will be unique; that is, if you want to make sure that all the male names will be different. The "Indexed" option enables the user to assign a particular order to list SmaName. FIG. 15.

Indexing is particular to Strings. Using indexing, if the user has an item where the stem reads "a lawyer asked his paralegal to call the client", and he/she wanted that to change in some other variant to "a doctor asked his nurse to call a patient". The user would never want to use "lawyer" and "patient" or "nurse" together, or "doctor" and "client" or "paralegal" together. Those three things need to stay together in the items and indexing permits the user to ensure that they do. The user can build sub-strings, so instead of just having String values like John or Mary, the user can have value subsets; for example, where lawyer, paralegal, client always go together and doctor, nurse, patient always go together. As shown in FIG. 16, the user has de-selected "Indexed" and left the remaining options alone.

CREATING AND IMPORTING STRING VALUES

Next the user must provide the available String values and the present invention provides several ways of doing that. One way is simply to click on the "Add" button in Fig. 16. Actually, in this preferred embodiment, everything on the icon bar of the "Create or Change Variable" dialog box of FIG. 15 (and for that matter everything on most of the other icon bars) can also be replicated with a right button click.

Another useful feature of this embodiment is the ability to save the male name String values (or any other list of String values) for subsequent use with different models. The "Exporting Strings" feature is used to save the list and then the "Importing Strings" feature is used to automatically populate the String values of the variable in the new model. Both ways are discussed below.

Using the "Add" Button

Click on the "Add" button in the "Create or Change Variable" dialog box shown in FIG. 16 and the system provides the user with a new dialog box: "Edit string SMaleName". FIG. 17. The user then enters a name, for example John, and then clicks "OK". This procedure is repeated, this time with the String value equal to Tom. FIG. 18. After several more male name String values are inputted the result looks like FIG. 19, where the list of male

names String values all appear, not surprisingly, in the "String values" window. The "Edit" button may be used to edit existing string values.

Using the "Export Strings" and "Import String" Buttons

5 The user can utilize the "Export Strings" function to save this particular list of String values for reuse with another model. The user clicks on the "Export Strings" button shown in FIG. 19 resulting in the "Export string to file" dialog box appearing. The user can then name the file the String values
10 will be saved in (e.g., "male_names") and save the file by clicking on the "Save" button in FIG. 20.

In this preferred embodiment, the file is saved into the directory "TCS/TCA/In. The saved Strings can be shared with other users. If the user needs "male_names" String values, all
15 he/she needs to do is use the "Import Strings" button (e.g., in FIG. 16) and choose the appropriate String file found in the resulting dialog box.

FIGS. 24 through 26 show parts of the above process for providing the String values for the String variables "SItems" and
20 "SFemaleName".

EDITING INTEGER VARIABLES

Select INum1 in the Variables window by making a left mouse button click on INum1 and then make a right mouse button click to open the options menu and "Edit". Once again the "Create or Change Variable" dialog box will appear. FIG. 21. In accordance with the naming convention and the auto-define variables function, "INum1" has been assigned the type "Integer". In the dialog box of FIG. 21 can be found the "Add to checksum" box discussed with respect to String variables and a new box entitled "Independent". FIG. 21. Checking the "Independent" box ensures that the value of the variable will fall into the range of values defined for that variable. If an independent variable (e.g., INum1 in FIG. 22) is not assigned a value in a constraint, a value from the defined range of values (shown in the FROM/TO/BY windows) is chosen at random. If an independent variable (e.g., INum1 in FIG. 22) is assigned a value in a constraint (e.g., "INum1 \neq INum2" (FIG. 49)), the value chosen for the independent variable will still fall within the defined range of values (shown in the FROM/TO/BY windows) chosen at random, but the actual value chosen for the variable will also be required to satisfy the constraint.

When "Independent" is checked a default range of from 1 (its lowest value) to 100 (its largest value) by 1 (or in increments of

1) appears. That is, at this point, INum1 can have the following values: 1,2,3,...,100. FIG. 22. As shown in FIG. 23, the range has been change to from 2 to 26 by 3, and therefore variable INum1 can have the following values: 2,5,8,...,26.

5 In a like manner, the range of values of independent integer "INum2" has been chosen by the user to be 2 to 27 by 5. FIG. 27. Finally, as IKey represents the answer, the user leaves the "Independent" box unchecked as its value will depend upon the values of the variables. FIGS. 28 and 29.

10 SPECIFYING THE CONSTRAINTS

To define relations for determining a value of "IKey" the user can click on "Add" button in the "Variation Constraints" window in FIG. 29. This will bring up the "Create or Change Constraints" dialog box. FIG. 30. This dialog box is divided
15 into three areas. The first entitled "Constraint" is the box wherein variables are constrained. The second comprises three work environments entitled "Operators", "Variables", and "Functions". Clicking on the title/tab activates the chosen
20 environment. Finally, there is a section entitled "Comment" which allows the user to provide comments documenting the particular constraint for use by the current user or another user at some later time.

Operators

The first tab in the box entitled "Operators" appears in bold indicating that the buttons associated with this tab are available to the user. That is, clicking on a button places its operator in the "Constraint" box. The buttons in the "Operators" environment appear in FIG. 30 and comprise mathematical operators: +, -, =, /, %, >, <, >=, <=, if, then, else, elseif, (), etc.

Variables

Clicking on the "Variables" tab, activates the "Variables" environment, and allows the user to enter into the "Constraint" box all currently identified variables. FIG. 31. The user just highlights the variable and clicks on the "Insert" button.

Functions

Clicking on the "Functions" tab, activates the "Functions" environment, and allows the user to enter into the "Constraint" box all TCA library functions. FIG. 32. The user just highlights the variable and clicks on the "Insert" button.

Moreover, clicking on a function brings the function to the top box of the "Functions" environment and provides the user with a definition of the function immediately below it. See, for example, FIG. 39.

Constraining "IKey"

The user clicks on the "Variables" tab and selects "Ikey".

FIG. 33. Clicking on the "Insert" button causes the system of the present invention to put "IKey" at the current cursor

location in the "Constraint" box. FIG. 34. Clicking on the

"Operators" tab and then clicking on the "=" operator button

results in the start of a constraint, namely: "Ikey=". FIG. 35.

Going back and forth from "Variables" to "Operators" and

inserting INum1, "+", and INum2 results in FIG. 36. The system

allows the direct entry of the constraint in the "Constraint"

box, however, providing the ability to pick and choose by

clicking facilitates constraint generation, by among other

things, reducing typographical errors.

Exporting and Importing Constraints

The user can export and import variation constraints in a

fashion similar to exporting and importing strings. The "Export

Constraints" and "Import Constraints" buttons in the "Model

Workshop" are used. FIG. 37. The only difference is that when

the user exports or imports constraints the system necessarily

exports/imports variables, variable constraints, and associated

comments. Clicking on the "Print Constraints" button, FIG. 37,

in one embodiment, prints out the constraints.

Constraining the Distractors

Multiple choice tests items are designed to have several responses from which the test taker will choose the correct answer. The correct answer is the key or in this example "IKey". The other responses are wrong answers or distractors and are called "IDistractor_". The user defines as many distractors as the test item calls for, in this example four (4) distractors. To add distractor constraints, the user can click the "Add" in "Distractor Constraints" window, FIG. 38, and the "Create or Change Constraints" dialog box will appear. FIG. 39.

To add an expression for distractor "IDistractor1", the user can either directly type it in the "Constraint" window/box or insert it by going to the "Functions" tab, selecting a function from the list, and inserting it by clicking on the "Insert" button. When a function is selected, a useful description of the function is displayed to the user. FIG. 39. Variables can be inserted into functions, for example, "INum1" and "INum2" into "min()". See FIGS. 39-41. Clicking on the "OK" button in the "Create or Change Constraints" dialog box finishes defining the constraint for the time being and all the constrained distractors appear in the "Distractor Constraints" window of the Model Workshop. See, e.g., FIG. 42.

Testing the Constraints

The systems permits the user to test the constraints by testing selected or all variables, variation (variable) constraints, and distractor constraints. The user checks off all the items to be tested and depending on the circumstances clicks on the "Test" buttons on the icon bar at the bottom of the appropriate Workshop window or the "Test All" button to the right of those windows. The ability to chose particular items to be tested is very helpful in locating the source of any problem.

FIG. 42. After TCA finishes testing all checked variables and constraints in FIG. 42, a "Test Result" window appears. FIG. 43. The "Test Result" for FIG. 42 was "Variable IDistractor4 is underconstrained!" In fact, it had not been constrained. After constraining IDistractor4 and clicking on the "Test All" button appears the next "Test Result". FIG. 44. This time TCA tells the user that the constraints "Looks Good!". If the constraints "Looks Good!" the next step is to click on Generate Variants" tab so as to be able to use TCA to automatically generate test item variants based on the model developed in the Model Workshop.

FIG. 45.

GENERATING TEST ITEM VARIANTS

To generate variants all the user needs to do is enter the number of variants to be generated in the "Number" box and click

on the "Generate" button. FIG. 45. The user can also adjust the "Prolog randomization". The generated variants will appear in the "Variants" window. In this case the system was requested to generate two (2) variants from family model "NEWMC\$R.DOC". The variants generated from this model have names NEWMC\$R1.doc and NEWMC\$R2.doc. That is, family model name variant number 1 and number 2. FIG. 46. Selecting a variant in the "Variants" window causes the variant to be displayed in the Microsoft® WORD window.

Note that at least three of the distractors for test item

NEWMC\$R1.doc equal the same value, namely, 6. Therefore, as initially constrained the distractors in this model can simultaneously have the same value, which is wrong. Therefore, the user will need to change the constraints to eliminate this possibility.

If based upon a review of the generated variants the user wishes to modify the constraints, he/she need only click on the "Model Workshop" tab. If the user does so, a warning appears to the effect that variants on tab 3 (the "Generate Variants" tab) will be deleted if not saved before changing the model. FIG. 47.

FIG. 48 shows part of the process of adding a new constraint in an attempt to resolve the distractor problem. The new constraint is that INum1 cannot equal INum2. Adding this constraint and then testing results in a "Looks Good". FIG. 49.

FIGS. 50 - 51 show the result of generating 2 variants from the new model. It appears that the distractor problem has been fixed.

WORKING WITH GENERATED VARIANTS AND GENERATING NEW MODELS

5 ACCEPTING VARIANTS

If the user is satisfied with one or more generated test item variants, the variants may be accepted. Selecting the variant "NEWMC\$R3.doc" and clicking on the "Accept" button in FIG. 52 leads to FIG. 53 where a dialog box entitled "Confirm" appears and the user is given one more chance to accept the variant or not. Since the user chose to accept the variant, it no longer appears in the "Variants" window of the "Generate Variants" tab. See FIG. 54.

DEFERRING AND DISCARDING VARIANTS

15 If the user does not "like" a variant, the variant can either be deferred by selecting and clicking on the "Defer" button, or discarded by selecting and clicking on the "Discard" button. FIG. 54. In deferring a variant, this preferred embodiment of TCA does not store the variant's check sum value, therefore, deferred variants could be generated again. On the other hand, discarded variant check sums are stored to ensure
20 that they will not be regenerated in the future.

CREATING NEW VARIANT MODELS FROM A GENERATED VARIANT

To create a new variant model (new children of the active model) from a particular variant, select the variant in the "Variants" window and click on the "Create Mdl." (Create Model) button on the icon bar located at the bottom of this window.

FIG. 54. A dialog box entitled "Confirm" will appear, the user clicks on the "Yes" button, FIG. 54, to create a new model. The new model creation is confirmed by a dialog box entitled "Model Created". FIG. 56. Thus confirming that the variant has been copied with a new name. The name of the new model appears in the "Model Created" dialog box. In this case, the new model is named "NEWMC\$RA.doc". The "R" means that the model is a root model, the "A" at the end of the name implies that it is a child of the model "NEWMC\$R.doc". FIG. 56. In this way, the variables and the constraints of the previous model are preserved, so the user does not have to go in and start variabilizing from the beginning.

ACCEPTED VARIANTS AND NEW FAMILY MODELS

If the user clicks on the "Family Overview" tab, FIG. 57, "NEWMC\$RA.doc" appears in the "Family members" window. A sun icon next to this name indicates that the model is active. The "snowflake" icon to "NEWMC\$R.doc" indicates that this model is "frozen". As soon as at least one item variant generated from an

item model is accepted, the item model is frozen. A frozen item model can be viewed and used to generate more item variables and unfrozen child models, but the model is blocked to future changes. Finally, the accepted variant "NEWMC\$R3.doc" appears in the "Accepted variants" window. FIG. 57.

WORKING WITH MODELS AND ACCEPTED VARIANTS

To begin working with the new model "NEWMC\$RA.doc" click on the name and the model will appear in Word® window. FIG. 58.

Click on "Set Attributes" button in FIG. 58 brings up the "Family Attributes" dialog box. FIG. 59. The user has the option of choosing either "Generic" or "Non-generic". Variants are considered "generic variants" if they are very familiar in terms of structure and content, otherwise variants are called "non-generic variants". The user also has the option of choosing the "Variant proximity". As can be seen in FIG. 60 the proximity can be "Near", "Medium" or "Far". Clicking "OK" after making the appropriate selections in the "Family Attributes" box associates the selections with the model.

To "extend" (make a copy) or remove a "frozen" model or "un-frozen" model, the user selects the item, right mouse-button clicks, and selects "Extend" or "Remove". FIG. 61. The same could be done by using buttons "Extend" and "Remove" that are

located on the bar at the bottom of the "Family members" window.

FIGS. 61 - 62.

EDITING THE PROFILE OF A VARIANT

5 To edit or copy the profile of an accepted variant, select the variant and click on the "Edit Profile" or "Copy Profile" button as appropriate. They are located on the bar at the bottom of "Accepted variants" window. FIG. 63. Clicking on "Edit Profile" brings up the "Profile of variant [name of variant]"

10 dialog box. FIG. 64. Clicking on the arrow next to the "Domain:" window in FIG. 64 brings down the selection of domains: Arithmetic, Algebra, Data Analysis, or Geometry. FIG. 65.

15 Clicking on the arrow next to the "Target template:" window in FIG. 64 brings down the selection of targets: "CBT" for computer based testing or "PPT" for paper and pencil testing. FIG. 66.

The user can also select either "Pure" or "Real" shown for example in FIG. 66. A Pure test item is one in a mathematical setting. A Real test item is one that is based on a real-life situation. The user can also select Route to TCS (ETS' Test Creation System) shown for example in FIG. 66. ETS' Test Creation System is disclosed in U.S. Patent No. 6,000,945, which is hereby fully incorporated by reference herein.

The GRE Difficulty Portion of the Profile of a Variant Window

The "GRE Difficulty" portion of the Profile of variant window shown in FIG. 67, for example, has several components. Clicking on the arrow next to the "Computation:" window, brings down the following selection which denotes the type of numbers used in the test item: Integers, Decimal/fractions, Radicals, or Not applicable. FIG. 68. In the same fashion, the "Cognition:" window provides selections for denoting the sort of process the test item requires the test taker to go through to arrive at the answer, namely: Procedural, Conceptual, or Higher order thinking. FIG. 69. The "Concept" window provides selections for denoting the basic concepts tested by the item, namely: Probability, Percent of a percent, Percent change, Linear inequality, or Not applicable. FIG. 70. "Adjust the slide to estimated difficulty:" allows the user to denote his or her opinion of the difficulty of the variant. See, for example, FIG. 70. The "Key:" window allows the user to denote the correct answer. Finally, the "Predicted Difficulty" or IRT b measure is calculated from other entries in the window (e.g., whether the item is arithmetic or algebra).

WORKING WITH FAMILY MEMBERS

Back at the "Family Overview" window, FIG. 71, by selecting the "frozen model" "NEWMC\$R.doc", right mouse-button clicking, and selecting "Extend", the user can extend the "NEWMC\$R.doc".

5 In response to selecting "Extend", the "Confirm" window pops up, FIG. 72 and clicking on "Yes" extends the model. New active model "NEWC\$RB.doc" then appears in the "Family members" window. FIG. 73.

Using a New Active Model to Generate Far Variants

10 The new model is immediately available for the user. Clicking on the Model Workshop tab for "NEWC\$RB.doc" brings the user to a place where he or she can modify the model. For example, the user could replace the stem:

"If SMaleName had INum1 SItems and SFemaleName had
15 INum2 SItems, how many SItems would they have
together?";

with "SStem"; add "SStem" as a new variable; and add

"If SMaleName had INum1 SItems and SFemaleName had
INum2 SItems, how many SItems would they have
20 together?";

as a String value for "SStem". See FIG. 73A. The user could also add other values for "SStem", for example,

"INum1 + INum2 = ?".

FIG. 73B. Thus, this preferred embodiment of the present invention allows the user to easily generate far variants (variants that are very different from one another) from this new stem by going to "Generate Variants", requesting 2 variants, FIG. 73D, and clicking on the "Generate" button. Thereby generating new variants "NEWMC\$RB3.doc", FIG. 73D and "NEWMC\$RB4.doc", FIG. 73E.

Creating Still More Models

Preferred embodiments of the present invention permit the user to make a child model from the "NEWC\$RB.doc" model of FIG. 73, that is, to extend it. The user selects "NEWC\$RB.doc", clicks on the "Extend" button, and then enters "Yes" in the "Confirm" window, FIG. 74. New active model "NEWC\$RBA.doc" appears. FIG. 75. Left button click to select this model, then right mouse button click to extend it or remove it. FIG. 75. To make a child model from new model "NEWC\$RA.doc", repeat the above procedure, New active model "NEWC\$RAA. doc" appears. The "AA" stands for a child of a child of the root model. See, FIGS. 76 - 77.

The constraints for any of the active models that are displayed in the "Family members" window can be changed. To change the constraints, select an active model in the "Family

members" window, click on tab Model Workshop, left button click to select a constraint, and then right button click to get the constraint option. FIG. 78.

PRINT OPTIONS

To print accepted variants click on tab "Family Overview" and then click on button "Print All" in FIG. 79. FIGS. 80A, 80B and 80C is the result. It is a print out of the variables and constraints for model "NEWMC\$R". Selecting the model "NEWMC\$RA" as the active model and in the Model Workshop clicking on the "Print Constraints" button in FIG. 81, results in a print out of the variables and constraints for model "NEWMC\$RA". See, FIGS. 82A, 82B, and 82C.

To print a model without constraints click on the "Family Overview" tab and select a model, for example NEWMC\$R.doc. In the Microsoft® WORD window entitled "NEWMC\$R.doc", select File and Print or just click on print button. The result is a print out of model "NEWMC\$R.doc" without constraints. See FIG. 83.

To print one of the accepted variants from a particular model, click on "Family Overview" tab, select a model, for example "NEWMC\$R.doc". In "Accepted variants" window, select one or more variants, for example "NEWMC\$R3.doc", "NEWMC\$R4.doc",

etc.. In the Microsoft® WORD portion for each variant print out the document. The test item variants appear in FIGS. 84 - 88.

GRE QUANTITATIVE COMPARISON ITEMS

5 To create a model for GRE Quantitative comparison items, the user starts as shown in FIGS. 1-3. However, instead of keeping Multiple choice as an item choice, "Quantitative comparision" is selected. FIG. 89. "Non-generic" and "Near" are also chosen is the example to be discussed. After saving this new family as
10 "NEWQC" the result is FIG. 90.

In the Microsoft® WORD document appears the title "TCA Quantitave Comparison Model"; there are also sections entitled "reserved for variants", "stem", "column A", and "column B". In the right part of the window you will see "Family Overview" tab
15 highlighted. In "Family members" you will see an icon with a sun and the name of the chosen variant, "NEWQC", next to it. The variant family name will have an extension "\$R.doc". The "sun" icon again indicates that the model is active. In the "Family members" window appear two highlighted buttons: "Extend" and
20 "Remove". These buttons enable the user to extend or remove the variant family, respectively. At the bottom of the "ETS Test Creation Assistant" window, you will see a toolbar with following titles: "Program -GRE", "Family -NEWQC\$R.doc", "Attributes - QC",

"Non generic", "Near", "Active Model . . . "NEWQC\$R.doc". FIG.
90.

FIG. 91 is a print out of "NEWQC\$R.doc". The idea of a QC
item is to compare values in columns A and B. FIG.91.

GMAT DATA SUFFICIENCY ITEMS

To create a model for GMAT Data Sufficiency items the
approach is generally the same as with the other item types
taking into account the concept behind the item type. See FIGS.
92 - 93.

FURTHER EXAMPLES OF ITEM MODELS

See FIGS. 94 - 106B for further examples of item models.

PROLOG SIMULTANEOUS CONSTRAINT SOLVER

Preferred embodiments of the present invention use PROLOG as its simultaneous constraint solver. Details of one preferred embodiment appear in the PROLOG SOURCE CODE APPENDIX. Brief
5 descriptions of the TCA Prolog (and Prolog-related) files are provided below.

HLP4lib.p4

This file provides a library of Prolog-predicates for use in solving mathematical constraints. For example, it provides a
10 Prolog predicate gcd(GCD, A, B) returns the GCD of two integers A and B. This file provides a library of Prolog IV accessory relations useful in high-level API.

PrlgExpr.l

This file provides the lexical specification for the Prolog-
15 expression scanner. Using this file, an appropriate scanner is generated. The scanner breaks the mathematical constraints into individual words and operators (called tokens) which are further used by the Prolog-expression parser.

PrlgExpr.y

0 This file provides the syntactic specification for the Prolog-expression parser. The file PrlgExpr.y provides the BNF-specification from which the parser-code is generated. While PrlgExpr.y is "almost" a parser, it contains, strictly speaking,

a specification to generate a parser. Therefore, using this file, an appropriate parser is generated. The parser accepts the tokens from the scanner, and recognizes the syntactic patterns of mathematical constraints (or parses mathematical constraints or computes a parse-structure for mathematical constraints). Having recognized the syntactic patterns, it transforms the mathematical constraints into appropriate Prolog clauses, and calls Prolog IV to solve those clauses.

hlp4API.h

This file provides a specification for API access to the mathematical constraints-solver, Prolog IV. Using this specification, other programs can access the constraints-solver to solve mathematical constraints.

TCA CONSTRAINT LANGUAGE

TCA uses a high-level language derived from Prolog and Algebra to write mathematical constraints. This section
5 describes the language and provides some example mathematical constraints.

The TCA constraint language is intended to help the test developers in writing models for math items to be cloned. As such, it is a language very similar to the test developers' language of choice: the mathematical notation to write algebraic
10 equations. It provides a suite of predefined high-level functions to choose from, high-level set operators (e.g. membership/iteration over a continuous range or a discrete set), and additional operators (e.g. and, or) to combine the
15 constraints in the desired fashion.

The TCA constraint language differs from procedural languages (e.g. C, Fortran) principally in that it is a goal-oriented language, that is users need specify only the constraints to be solved, and not how to solve them. In
20 addition, the TCA constraint language has (almost) no program-flow control mechanisms (e.g., no goto's, no while loop). Program-flow is controlled by the constraint-solver. Further, as expected from a mathematical constraint-solver, it is

constraint-order independent (e.g. $X = 2$, $Y = X + 2$. can equally well be written as: $Y = X + 2$, $X = 2$.).

NOTATIONAL CONVENTION

Solutions follow the arrow (\Rightarrow) after the constraint.

Item* represents 0 or more instances of Item.

Item+ represents 1 or more instances of Item.

Item? represents 0 or 1 instance of Item.

func/n represents a function with n arguments, for example, max/2 represents the function max with two arguments (e.g., max(4, 7)), while max/1 represents the function max with 1 argument (e.g. max([4, 7])).

In describing the arguments to a function, the notation +Arg is used to indicate that the value of the Arg must be given (i.e., Arg is an input argument), and

-Arg to indicate that the value of the given Arg is set by the function (i.e., Arg is an output argument). A simple

Arg (without + or -) indicates that one can use the argument with input or output parameter.

TCA CONSTRAINT LANGUAGE IN DETAIL

The TCA constraint language (often referred to herein as the language) is syntactically based on conventional mathematical notation system, with some additional syntactic constructs to combine multiple constraints in various ways (e.g., conjunction,

disjunction, if-then-else) and to supply some low-level details often hidden in mathematical notations (e.g., type, precision, stepsize for iteration). Note that the TCA constraint language is case-sensitive; e.g., Xvar is different from xvar.

5 The TCA constraint solver can solve linear constraints and a large class of nonlinear constraints. The user need specify only the constraints to be solved, and not how to solve them. The TCA returns all the solutions to the specified constraints. Further, all the constraints and functions are relations which have

10 minimal input/output directional restrictions; i.e., one can use the same argument in a function to provide a known parameter and to compute an unknown value. For example, one can use the same constraint $Z*Z = X*X + Y*Y$ in multiple ways:

 Given $X (:3)$ and $Y (:4)$, to compute the value of Z ; i.e.:

15 $X = 3, Y = 4, Z*Z = X*X + Y*Y. \Rightarrow \{Z: 5; Z: -5\}.$

 Given $Z (:5)$, to compute the possible values for X and Y ;

 i.e.: $5*5 = X*X + Y*Y, \text{int}(X, Y). \Rightarrow \{X: 3, Y: 4; X: 4, Y: 3\}.$

 Given $Z (:5)$ and $X (:3)$, compute the value of Y ; i.e.: $5*5 = 3*3 + Y*Y. \Rightarrow \{Y: 4\}.$

20

The constraints in the language are specified using a combination of the representational devices offered by the language: basic

elements, type specifications, set specification, algebraic constraints, logical combinations.

BASIC ELEMENTS

Basic elements are the basic objects of the language. These elements are later referred, in general, as the terms.

1. Constants.

a) Numbers: 4, 10, -4, 5.6, -6.7.

b) Symbolic number: pi.

c) Symbolic constants (atom): Symbolic constant-names must start with a lowercase letter. For example, countVar, accountant. Any alphanumeric symbol that starts with a lowercase letter is considered a constant. Symbolic constants may not occur in an algebraic constraints: "X = 5 + accountant" is invalid.

2. Variables.

Variable-names must start with an uppercase letter: X, Xin, Yout. Note that case (uppercase, lowercase) is important. For example, X is a variable whereas x is a constant.

A special variable, called anonymous variable, may also be used. An anonymous variable is written as: "_". An anonymous variable acts like a placeholder, as a don't-care variable. Each anonymous variable, even when used multiple times within the same constraint, is distinct and one may not refer to it. For example, function divmod(N, D) returns a list: [N div D, N mod

for a function-call is: function-name(args*). For example,
abs(X) or mean(A, B) or median([1,4,3,7, 9]).

Note that the functions return values of the types indicated
below. The functions are valid only when their return-values are
in the appropriate context. That is, functions returning numbers
must be in the context of the algebraic expressions (e.g.

(assuming X is a variable of type number): X= sqrt(25));

functions returning lists must be in the context of the list
expressions (e.g. (assuming L is a variable of type list) : L=

sort([1,7,5,3,2]); functions returning symbols must be in the
context of the symbol expressions (e.g. even(4)= true.).

The predefined functions for one preferred embodiment are listed
below with the following notational convention:

The return-type of a function is indicated as: =>

Return-Type.

Names of the function-arguments of type integer start
with I;

Names of the function-arguments of type real start with
R;

Names of the function-arguments of type number start
with N;

Names of the function-arguments of type
lists-containing-numbers start with NL;

Names of the function-arguments of type list
(containing any kind of elements) start with L).

The predefined functions include:

`max(+N1, +N2)` (\Rightarrow Number) Returns the maximum of
5 numbers N1 and N2 e.g. `X= max(4, -5)`.

`max(+NList)` (\Rightarrow Number) Returns the maximum of
a list of numbers e.g. `X= max([2, sqrt(9)])`.

`min(+N1, +N2)` (\Rightarrow Number) Returns the minimum of
numbers N1 and N2 e.g. `X= min(4, -5)`.

10 `min(+NList)` (\Rightarrow Number) Returns the minimum of a
list of numbers e.g. `X= min([2, sqrt(9), 2^2])`.

`mean(+N1, +N2)` (\Rightarrow Number) Returns the mean of numbers
N1 and N2 e.g. `X= mean(4, sqrt(9))`.

15 `Mean(+NList)` (\Rightarrow Number) Returns the mean of a list
of numbers e.g. `X= mean([2, sqrt(9), 2^2])`.

`median(+NList)` (\Rightarrow Number) Returns the median of a list
of numbers e.g. `X= median([2, sqrt(9), -4, 7])`.

`gcd(+IN1, +IN2)` (\Rightarrow Integer) Returns the gcd of integers
IN1 and IN2 e.g. `X= gcd(4, 6)`.

20 `lcm(+IN1, +IN2)` (\Rightarrow Integer) Returns the lcm of integers
IN1 and IN2 e.g. `X= lcm(4, 6)`.

`sqrt(N)` (\Rightarrow Number) Returns the positive
square-root of (positive number) N e.g. `X= sqrt(20)`.

`cubert(N)` (\Rightarrow Number) Returns the cube-root of
(positive number) N e.g. `X= cubert(20)`.

`reverse(+List)` (\Rightarrow List) Returns the reversed List e.g.
`X= reverse([1,2,3,7,5])`.

5 `sort(+NList)` (\Rightarrow List) Returns the list of numbers
sorted in numerically ascending order e.g. `X=`
`sort([1,2,5,3])`.

`permute(+List)` (\Rightarrow List) Returns the various
permutations of the given List e.g. `X= permute([1,2,3])`.

10 `select_r_of_n_ordered(IN, IR)` (\Rightarrow Integer) Returns no.
of ordered subsets of size IR from the set of size IN.

`select_r_of_n (IN, IR)` (\Rightarrow Integer) Returns no. of
unordered subsets of size IR from the set of size IN.

15 `random()` (\Rightarrow Integer) Returns a randomly
generated integer e.g. `X= random()`.

`random(+List)` (\Rightarrow Same-as-the-Given-List-Element)
Returns a random element from the given list e.g. `X=`
`random([a,2.5,6,c])`.

20 `random(+IMax)` (\Rightarrow Integer) Returns a random integer
between 1 through (positive integer) IMax e.g. `X= random(7)`.

`even(+IN)` (\Rightarrow true) Returns literal constant true
if IN is an even integer e.g. `even(10)= true`.

odd(+IN) (=> true) Returns literal constant true
if IN is an odd integer e.g. odd(11)= true.

is_perfect_square(+IN) (=> true) Returns literal
constant true if IN is a perfect square e.g.
perfect_square(16)= true.

isnot_perfect_square(+IN) (=> true) Returns literal
constant true if IN is not a perfect square e.g.
isnot_perfect_square(17)= true.

is_perfect_cube(+IN) (=> true) Returns literal
constant true if IN is a perfect cube e.g.
perfect_square(64)= true.

isnot_perfect_cube(+IN) (=> true) Returns literal
constant true if IN is not a perfect cube e.g.
isnot_perfect_square(25)= true.

is_prime(+IN) (=> true) Returns literal constant true
if IN is a prime integer e.g. is_prime(11)= true.

isnot_prime(+IN) (=> true) Returns literal constant
true if IN is not a prime integer e.g. isnot_prime(10)=
true.

5. Algebraic Expressions (referred to as: AlgExpr).

Expressions can be built by combining other algebraic
expressions (e.g. numbers, variables, functions) using the
arithmetic operators. Valid operations include:

"+" (addition): AlgExpr + AlgExpr, e.g. A + B, or 4 + C, or sqrt(16)+7.

"-" (subtraction): AlgExpr - AlgExpr, e.g. A - B, or 4 - C, or sqrt(16)-7.

5 "Unary -" (unary negation): - AlgExpr, e.g. -B, or -7.

"*" (multiplication): AlgExpr * AlgExpr, e.g. A * B, or 4 * C, or sqrt(16) *7.

10 "/" (division): AlgExpr / AlgExpr, e.g. A / B, or 4 / C, or sqrt(16) /7.

"%" (modulo): AlgExpr % AlgExpr, e.g. A % B, or 4 % C, or sqrt(16) %7.

"\" (quotient): AlgExpr \ AlgExpr, e.g. A \ B, or 4 \ C, or sqrt(16) \7.

15 "^" (exponentiation): AlgExpr ^ AlgExpr, e.g. A ^ B, or 4 ^ C, or sqrt(16) ^7.

"!" (factorial): AlgExpr!, e.g. A!, or 4!.

"|" (abs): | AlgExpr |, e.g. |A - 10|, or | sqrt(16) -7|.

20 The precedencies of the arithmetic operators are as follows

(higher precedence operators are specified before the lower

precedence once): !; ^; - (unary negation); % & \; /; *; + & -

(subtraction); | ... |.

CONSTRAINT SPECIFICATION

1. Type Constraint Specification.

These constraints specify the type of the variables in the constraint. The default variable type is "real".

5 "int(VarLst)" e.g.: int(X), int(X, Y).

"real(VarLst)" e.g.: real(Y), real(X, Y).

"fraction(VarLst)" e.g.: fraction(X), fraction(Y, Z).

"symbol(VarLst)" e.g.: symbol(X), symbol(Y, Z).

"list(VarLst)" e.g.: list(X), list(X, Y).

10 "ongrid(Var)" Specifies that the value of the given
variable must be an integral multiple of
its precision. [This is the default
behavior of the variables.]

e.g.: ongrid(X).

15 "offgrid(Var)" Specifies that the value of the given
variable need not be an integral
multiple of its precision.

e.g.: offgrid(X).

2. Optimizable-Relation Specification.

20 A user may help the constraint-solving process by
identifying the optimizable relations. Such relations must be
valid for the entire constraint (i.e., they may not be a part of
a disjunctive clause), and they may be moved around by the

constraint-solver without any logical incorrectness, usually to the start of the clause.

The devices to identify such relations include:

"eq_vars(Vars)" Variables in the given
comma-separated- list must be
equal to each other.

e.g.: eq_vars(X, Y, Z).

"neq_vars(Vars)" Variables in the given
comma-separated- list may not
be equal to each other.

e.g.: neq_vars(X, Y, Z).

"neq_varvals(Var, Vals)" The given variable may not be
equal to any of the values
specified in the
comma-separated-list.

e.g.: neq_varvals(X, 2, 5, 7).

"optimizable_rel(Relation)" The specified Relation is
optimizable. This is a
generalization of the special
optimizations presented above.

e.g.: optimizable_rel(X/=5).

Note that neq_vars(X,Y) (and, similarly, eq_vars(X,Y)), while semantically equivalent to $X \neq Y$, is operationally different from

$X \neq Y$ in that the constraint-solver optimizes such declarations (which it cannot do in case of declarations such as: $X \neq Y$ or $X=Y$).

3. Precision Specification.

Constraints are solved with certain precision. The default precision is: 0.01 for reals, 1 for integers. Nonlinear constraints are solved by enumerating through the potential solution-interval. A variable may assume a value which must be an integer multiple of its precision.

Precision for a variable can be changed individually by using the following construct for the variable in the type-specification: {Var, Precision}. For example, `real({X, 0.02})` specifies a precision of 0.02.

4. Relational Constraints (RelExpr).

Valid relational constraints (and the associated relational operators) include:

"=" (equality): AlgExpr = AlgExpr, e.g. $A = B$,
 $A + 5 * Z = \text{sqrt}(16) + C$.

"!=" (inequality): AlgExpr != AlgExpr, e.g. $4 * X +$
 $5 * Y \neq 2 * A + 3 * B$.

"<" (less than): AlgExpr < AlgExpr, e.g. $A < B$, $A +$
 $4 * Z < 4 + C$.

">" (greater than): AlgExpr > AlgExpr, e.g. $A > B$, $A + 4 * Z > 4 + C$.

"<=" (less than or equal to): AlgExpr <= AlgExpr e.g. $A <= B$, $A + 4 * Z <= 4 + C$.

5 ">=" (greater than or equal to): AlgExpr >= AlgExpr, e.g. $A >= B$, $A + 4 * Z >= 4 + C$.

"NOT": NOT RelExpr, e.g. NOT(is_prime(X) = true).

5. Ranges.

10 "Continuous interval": To specify a variable Var within a continuous range: [LowerExpr RL Var RL UpperExpr] where RL is one of the relational operators: { <, <=, >, >= }, e.g. to specify the constraint that X may range from 0 through 10: $[0 \leq X \leq 10]$.

15 "Continuous exterval": To specify a variable Var outside a continuous range: [! LowerExpr RL Var RL UpperExpr] where (RL is one of the relational operators: { <, <=, >, >= }.) For example, to specify the constraint that X must be outside the range from 0 through 10: $[! 0 \leq X \leq 10]$.

20 "Discrete inclusive range": To specify a variable Var within an enumerated range: Var in [Expr1, Expr1, ...]. For example, X in $[1, 5, 4 + 3, a, Y^2, 15]$, or, X in $[1, 2 * Y + Z, 7, Z]$. The Var assumes each value from the given set of values.

"Discrete exclusive range": To specify a variable Var outside an enumerated range: Var notin [Expr1, Expr2, ...]. For example, X notin [1, 5, 4+3, a, Y^ 2, 15], or, X notin [1, 2*Y+Z, 7, Z].

6. Enumerated Ranges.

To specify an enumerated range, we use: [LowerExpr RL Var RL UpperExpr step Expr] where (RL is one of the relational operators: { <, <=, >, >= }). Thus, for example, to specify that (a real variable) X enumerates through the range from 7 though 10 by a step of 0.5: [7<= X<= 10 step 0.5]. The example constraint is equivalent to: (X= 7; X= 7.5; X= 8; X= 8.5; X= 9; X= 9.5; X= 10).

To specify a discrete enumerated range, we use: Var from List. Thus, for example, to specify that (a variable) X can take one of the values from the set [a, b, c, d], we use: X from [a, b, c, d]. Similarly, to specify that another variable Y can take a value from the set of primes between 2 and 10, we can use: Y from [2, 3, 5, 7] .

If a continuous enumeration range is not closed on lower [upper] side (e.g., left relational operator is <), the lower [upper] expression is incremented [decremented] by the variable-precision for expanding the enumeration range. Thus, for example, for a real variable X with precision of 0.1, [7< X<

10 step 0.5] . => (X= 7.1; X= 7.6; X= 8.1; X= 8.6; X= 9.1; X= 9.6).

Note that an enumerated range is different from a regular (i.e. non-enumerated) range. The difference is especially visible for open (or, partially closed) ranges. For example, for a real variable X with precision of 0.1, the constraint: [7< X< 10 step 0.5], X= 7.4. is not successful, but the constraint: [7< X< 10], X= 7.4. succeeds.

Further, the main-variable in an enumerated range is considered independent, whereas the main-variable in a non-enumerated range is not independent. This difference becomes important when one is trying to generate different solutions for a constraint from the solver. While solving for different solutions, the constraint-solver tries to find different values for the independent variables in different solutions. It makes no such effort for non-independent variables.

Note that a user may not use variables in specifying the boundary(s) of an enumerated range when solving the constraints in the unique-order. As such, when solving the constraints in unique order, an enumeration range such as: [C+1 <=X <= C+3 step 1] is not acceptable because it uses variables in specifying the boundaries of the enumerated range.

7. if-then-else Constraint.

if (Condition) then (Then-Constraint) else
 (Else-Constraint). For example, if (is_prime(X)= true) then (Y=
 found_an_x_as_prime) else (Y= no_x_is_prime). (if-then alone
 also be used e.g. if (X== 5) then (Y= 7).)

Note that the semantics of the if-then-else constraint is:
 if Condition is ever true, then only the Then-Constraint is
 tested (i.e. executed). Only when the Condition is never true,
 the Else-Condition is tested (i.e. executed). Thus, for example,
 the following if-then-else constraint produces the results shown

below:

```
int(X), [1<=X<=4 step 1], if (even(X)= true)
then Y= x_is_even else Y= x_is_odd.    =>
X: 2, Y: x_is_even;
X: 4, Y: x_is_even.
```

Refer to the if-then-elseif constraint for a slightly different
 kind of constraint.

8. if-then-elseif Constraint.

if (Then-Condition) then (Then-Constraint) elseif
 (Else-Condition) then (Else-Constraint). e.g. the following
 constraint produces the absolute-values of X:

```
if (X>= 0) then (Y= X) elseif (X< 0) then (Y= -X).
```

Note that the semantics of the if-then-elseif constraint is:
 if Then-Condition is true, then the Then-Constraint is tested

(i.e. executed); or if Else-Condition is true, then the Else-Constraint is tested (i.e. executed). Thus, for example, the following if-then-elseif constraint produces the results shown:

```

5      int(X), [1<=X<=5 step 1], if (even(X)= true)
      then Y= x_is_even elseif (odd(X)= true) then
      Y= x_is_odd.    =>
          X: 1, Y: x_is_odd;
          X: 2, Y: x_is_even;
10         X: 3, Y: x_is_odd;
          X: 4, Y: x_is_even.

```

Refer to the if-then-else constraint for a slightly different kind of constraint.

9. Freeze Constraint.

15 Usually, one is interested in exploring the entire solution-space. However, there are times when one is satisfied with the solution (set) received so far, and wishes to freeze the solution (set) discovered so far. The freeze constraint is represented by the keyword: freeze.

20 10. Primitive succeed and fail constraints.

One can force a constraint to fail by conjuncting fail with it. Thus, for example, X= 4, fail. => false. Similarly,

succeed succeeds vacuously e.g. if $(X > 4)$ then succeed else fail.

11. Period (.) at the end of constraints.

12. Combining Constraints.

5 Grouping constraints together: "(Constraint)".

For example, $(X = 4 + X, Y = 2)$.

Conjunction (and): "Constraint1 , Constraint2".

For example, $X * X = 25, Y = 5, X = Y. \Rightarrow \{X: 5, Y: 5\}$.

Disjunction (or): "Constraint1 ; Constraint2".

10 For example, $X * X = 25; Y = 5, X = Y. \Rightarrow \{X: 5; X: -5; X: 5, Y: 5\}$.

Negation: "NOT Constraint".

For example, $\text{NOT}(X = 5)$.

Note that $\text{NOT}(X = 5)$ (i.e., it is never the case that X is 5) is not equivalent to $X \neq 5$ (i.e. the case when X is not 5). Thus:

15 X in $[1, 2], X \neq 1$. produces only one answer: $X: 2$ (because $X \neq 1$ succeeds when $X: 2$), whereas: X in $[1, 2], \text{NOT}(X = 1)$. fails (because $X = 1$ succeeds when $X: 1$).

WRITING CONSTRAINTS IN TCA CONSTRAINT LANGUAGE

20 The TCA constraint language combines the algebraic language with the syntax and semantics of logic programming language (for example, Prolog). It differs considerably from the procedural programming languages (for example, C, Fortran) which rely on

program-flow control to specify a procedure to solve the problem at hand. Major differences between procedural languages and the TCA constraint language of the present invention include:

TCA constraint language is declarative: In the TCA
5 constraint language, one specifies only the constraints to be solved, not how to solve them.

Constraints are order-independent: In the TCA constraint language, the constraints are order-independent e.g. $X = 2$, $Y = X + 2$. is the same as: $Y = X + 2$, $X = 2$. In procedural languages,
10 statements are order-dependent, e.g. in C, $X = 2$; $Y = X + 2$; is different from: $Y = X + 2$; $X = 2$. An exception to the order-independence of rules is the case where we use the continuous-range constraint (e.g., $[2 \leq X \leq 5]$) for integer variables and invoke functions with the variable from the
15 continuous range (e.g., $Z = \text{gcd}(X, Y)$). In such situations, the solver's behavior depends on the relative position of the variable-type-declaration (e.g., $\text{int}(X)$) vs. the continuous range declaration (e.g., $[2 \leq X \leq 5]$). In general, the user of the present invention should put the variable-type-declaration as
20 soon as he/she knows it. For example, $\text{int}(X)$, $Y = 5$, $[2 \leq X \leq 10]$, $Z = \text{gcd}(X, Y)$.

Constraints are solved all at once as a whole.

TCA constraint language provides its own program-flow control. As such, the TCA constraint language provides (almost) no other program-flow control mechanism. Procedural languages, on the other hand, have to provide a large number of them. For
5 example, goto, while, if-then-else, and the implicit left-to-right, top-to-bottom program-flow.

TCA constraints are (mostly) bidirectional. Because the TCA constraint language uses relations to implement most functions and operators, one can use the same function [operator] to map a
10 set of arguments to a result, or to map a result back to a set of arguments. Thus, for example, $X = 5! \Rightarrow X: 120$. Further, $120 = N! \Rightarrow N: 5$. In procedural languages, one has to explicitly apply the reverse function to achieve the effect illustrated above. For example, in C programming language, $X = \text{factorial}(5);$
15 $\Rightarrow X = 120;$ and $Y = \text{reverse_factorial}(120); \Rightarrow Y = 5$.

TCA constraint language has no value-storage and no assignment.

With these fundamental differences between the logical and procedural paradigms, the techniques to achieve solutions are
20 also different. In the following section, we describe some of the techniques to write the constraints and to solve them in TCA.

SOME TECHNIQUES TO SOLVE CONSTRAINTS IN TCA

1. Variable Type Specification.

It helps if you can identify the type of the variable explicitly, particularly if it is not real (e.g. integer).

Examples of such type-specification follow:

`square(R) = 100, int(R).`

`X=Y^3, X=8, int(Y).`

`X=Y^3, X=8, real(Y).`

2. Range specification.

One can specify boundary conditions for a variable using any of the relational operators (e.g. `>`, `>=`, `<`, `<=`) available. One can also specify the boundary using the range or the discrete set notation. Some examples of boundary specification follow:

`X= 2, [4< Y< 5*X], Z=Y+3.`

`[2^2 <= X<= 2^4], Y= X*2.`

3. Enumerated-Range Specification.

One can specify an enumerated range for a variable using the enumerated range construct. Some examples of enumerated-range specification follow:

`X= 2, [4< Y< 5*X step 0.5], Z=Y+3.`

`[2^2 <= X<= 2^4 step 0.3], Y= X*2.`

4. Efficient Solving.

For performance reasons, it is desirable to use only the constants in the enumerated range specifications of the

independent variables, and impose any other constraints later in the constraint. Thus, for example, the following constraint:

```
int(X,Y,Z), [1<=X<= 100 step 1], [1<=Y<=100
step 1], [gcd(X,Y)<=Z<=100 step 1];
```

5 can be solved more efficiently as:

```
int(X,Y,Z), [1<=X<= 100 step 1], [1<=Y<=100
step 1], [1<=Z<=100 step 1], Z>= gcd(X,Y).
```

5. Representing lists and tables.

One can use (multilayered i.e. lists containing lists) lists
10 to represent tables. For example, a 2x3 table (i.e. a table with
2 rows and 3 columns) can be represented as a 2-element list of
3-element lists e.g. Table_2_3= [[1, 5, 7], [10, 50, 70]]. One
can access the various table-elements using the (potentially,
multi-indexed) list-element-access notation: ListName [Index1 [,
15 Index2 [, Index3 ...]]]. Note that the first element of the
list is indexed 1, second element is indexed 2, and so on.
Multiple indices are used for multilayered lists.

Some constraints involving tables and table-elements and
their solutions are shown below:

```
20 Tbl_4_1= [1, 3, 5, 7], X= Tbl_4_1[2].      =>   X: 3.
Tbl_3_2=[[1,3],[5,7],[9,11]], X=Tbl_3_2[2].  =>   X: [5, 7].
Tbl_3_2=[[1,3],[5,7],[9,11]],X=Tbl_3_2[2,1]. =>   X: 5.
Tbl_3_2=[[a,b],[c,d],[e,f]], X=Tbl_3_2[2,2]. =>   X: d.
```

6. Bidirectionality of functions and operators.

Since the operators and (a large number of) functions in TCA are implemented using relations, one can use the same operators and the functions to map forward and backward and a mixture of forward-and-backward mappings. For example, $X = 4! + 5. \Rightarrow X: 29$. On the other hand, $29 = N! + C, C > 0. \Rightarrow (N: 4, C: 5; N: 3, C: 23; N: 2, C: 27; N: 1, C: 28; N: 0, C: 28)$. Similarly, $29 = N! + 5. \Rightarrow N: 4$.

7. Constraints are Solved in Order-independent Fashion.

Because the constraints are solved as a whole, solutions to the constraints in TCA are (mostly) independent of the constraint-order. Thus, the constraints: $Y = X^2, Y = 2 * Z, Z = 2$. and $Z = 2, Y = 2 * Z, Y = X^2$. provide exactly the same set of solutions: $(X: 2, Y: 4, Z: 2; X: -2, Y: 4, Z: 2)$.

As a practical matter, though, since the constraints are solved left-to-right by the constraint-solver, it often helps to write the constraints in an order such that the more determined constraints are to the left of the less determined constraints.

8. Constraints are Solved as a Whole.

All the constraints specified for one constraint are all solved as a whole, and not partially. This is particularly important in the case of the TCA where constraints are entered on different lines without any explicit operators (e.g. comma or

semicolon) combining them (TCA supplies the default comma-operator (i.e. conjunct) between adjacent constraints) and thus one might get the incorrect impression that the constraints are solved independently.

9. Variable Names are the Links to Bind Various Constraints.

One binds various constraints through the variables used in them. Thus, use of the same variable X in constraints $C1$ and $C2$ (when $C1$ and $C2$ are joined together by the and (i.e. comma) operator) is a statement to solve the constraints $C1$ and $C2$ to find the common value for X . For example, $5*5 = X*X + Y*Y$, $\text{int}(X, Y)$, $X = \text{cubert}(27)$. $\Rightarrow \{X: 3, Y: 4\}$. solves the constraints $5*5 = X*X + Y*Y$, $\text{int}(X, Y)$ and $X = \text{cubert}(27)$ to provide the common solution: $\{X: 3, Y: 4\}$, and discards the other solution: $\{X: 4, Y: 3\}$ for the first constraint.

As a corollary, using the same variable-name in multiple constraints forces them to find a common solution. That is, you may unintentionally restrict a solution space by unintentionally using the same variable name across multiple constraints.

10. Use of Sets and Ranges.

One can use sets and ranges to solve constraints over continuous ranges or discrete sets. For example, $[1 \leq X \leq 10 \text{ step } 1]$, $Y = X*X$, $\text{int}(X, Y)$. returns (in Y) squares of integers

from 1 through 10. Similarly, X in [-2, -4, 2, 4], Y= X*X*X.
returns (in Y) the cubes of numbers from the given set. Sets and
ranges can often be used in situations which might require
loop-operators in procedural languages.

11. Logical Operators.

One can use conjuncts (the comma operator: ,), disjuncts
(the semicolon operator: ;), negation (NOT), and their
combination (using parentheses) to provide any needed
program-flow control.

12. Equality by Assigning Same Variable Name.

One can impose equality constraint on variables by
explicitly equating them or by just naming them as the same
variable. By corollary, variables with the same name must have
identical value. Thus, for example, [Div1, Mod1]= divmod(16, 3),
[Div2, Mod2]= divmod(17, 3), Div1= Div2. => (Div1: 5, Div2: 5,
Mod1: 1, Mod2: 2). We can impose the same constraint with more
clarity and brevity as: [Div, _] = divmod(16, 3), [Div, _]=
divmod(16, 3). => (Div: 5).

Further descriptions of preferred embodiments of the present
invention appears in the Figures and both Source Code Appendices,
all of which are hereby incorporated herein in full.

VISUAL BASIC SOURCE CODE APPENDIX

TABLE OF CONTENTS¹

5	TCA.vbp	VBSCA -1-
	AXProlog.vbp	VBSCA -4-
	Common.bas	VBSCA -5-
10	Main.bas	VBSCA -6-
	modUtil.bas	VBSCA -7-
15	MTAPI.BAS	VBSCA -12-
	MTDeclaration.bas	VBSCA -17-
	MTUtil.bas	VBSCA -21-
20	Timer.bas	VBSCA -28-
	Constraint.frm	VBSCA -29-
	EditConstraint.frm	VBSCA -50-
25	Form1.frm	VBSCA -52-
	frmAbout.frm	VBSCA -54-
30	frmAttributes.frm	VBSCA -55-
	frmComments.frm	VBSCA -60-
	frmDifficulty.frm	VBSCA -62-
35	frmDrag.frm	VBSCA -76-
	frmIED.frm	VBSCA -79-
40	frmIndexedString.frm	VBSCA -81-

¹ All software COPYRIGHT 1999 ETS except for MTAPI.BAS

	frmNew.frm	VBSCA -89-
	frmNewModel.frm	VBSCA -94-
5	frmProgram.frm	VBSCA -97-
	frmProgress.frm	VBSCA -100-
	frmProlog.frm	VBSCA -102-
10	frmSplash.frm	VBSCA -104-
	SetPrecision.frm	VBSCA -108-
15	String.frm	VBSCA -111-
	TCA.FRM	VBSCA -114-
	Variable.frm	VBSCA -217-
20	Application.cls	VBSCA -254-
	CCLones.cls	VBSCA -255-
25	CConstraints.cls	VBSCA -261-
	Checksum.cls	VBSCA -268-
	Clone.cls	VBSCA -270-
30	CModels.cls	VBSCA -279-
	Constraint.cls	VBSCA -283-
35	ConstraintSolver.cls	VBSCA -288-
	CVariables.cls	VBSCA -297-
	CVariants.cls	VBSCA -308-
40	DifficultyEstimate.cls	VBSCA -311-
	DocStatus.cls	VBSCA -313-
45	DSMODEL.CLS	VBSCA -314-

	Family.cls	VBSCA -322-
	File.cls	VBSCA -328-
5	FileFind.cls	VBSCA -333-
	GMATDifficultyEstimate.cls	VBSCA -336-
10	GREDifficultyEstimate.cls	VBSCA -340-
	IniFile.cls	VBSCA -345-
	LockedItem.cls	VBSCA -350-
15	Model.cls	VBSCA -362-
	PrintModel.cls	VBSCA -381-
20	Progress.cls	VBSCA -384-
	Prolog.cls	VBSCA -386-
	PSMODEL.cls	VBSCA -392-
25	QCModel.cls	VBSCA -403-
	StringSolver.cls	VBSCA -410-
30	StringSolverx.cls	VBSCA -412-
	SubString.cls	VBSCA -413-
	Value.cls	VBSCA -417-
35	VarFraction.cls	VBSCA -419-
	Variable.cls	VBSCA -428-
40	VarInteger.cls	VBSCA -432-
	VarReal.cls	VBSCA -439-
	VarString.cls	VBSCA -449-
45	VarUntyped.cls	VBSCA -456-

Win32API.cls VBSCA -462-
Word.cls VBSCA -466-

PROLOG SOURCE CODE APPENDIX

TABLE OF CONTENTS²

```
HLP4lib.p4 . . . . . PROLOG SCA -1-
PrlgExpr.l . . . . . PROLOG SCA -13-
PrlgExpr.y . . . . . PROLOG SCA -16-
hlp4API.h . . . . . PROLOG SCA -90-
```

CLAIMS

WE CLAIM:

1. A computerized method for creating test item models and generating test item variants comprising the steps of:
 - 5 a. obtaining a test item;
 - b. creating a model by;
 - i. identifying elements of the test item to be variabilized;
 - ii. variabilizing the elements to create variables;
 - 10 iii. defining the variables;
 - c. generating a test item variant using a simultaneous constraint solver.
2. The method according to claim 1, wherein said model creation further comprises specifying constraints that define the
15 relationship among the variables.
3. The method according to claim 2 further comprising the step of accepting and retrievably storing the test item variant.
4. The method according to claim 3 further comprising the step of accepting and retrievably storing the test item model.
- 20 5. A computerized method for generating test item variants, the method comprising:
 - b. identifying elements of a test item or a test item model to be variabilized;
 - c. variablizing the identified elements;

- d. defining the variables;
- e. specifying constraints;
- f. using a simultaneous constraint solver to determine values for the variables;
- 5 g. generating test item variants.

6. A computerized system for generating test item variants from test item models comprising:

- a. means for retrievably storing test item models;
- b. means for selecting a test item model;
- 10 c. means for simultaneously solving test item model constraints;
- d. means for generating test item solutions by simultaneously solving test item model constraints;
- e. means for displaying, accepting and retrievably storing valid test item solutions.
- 15

7. The computerized system of claim 6 further comprising:

- a. means for obtaining a test item;
- b. means for identifying elements of the test item to be variabilized;
- 20 c. means for variabilizing the elements to create variables;
- d. means for defining the variables;

e. means for accepting the variabilized test item with defined variables as a test item model.

8. The computerized system of claim 7 further comprising means for specifying constraints that define the relationship among he variables.

9. The computerized system of claim 8 further comprising implementation of the VISUAL BASIC SOURCE CODE set forth in the VISUAL BASIC SOURCE CODE APPENDIX.

10. The computerized system of claim 8 further comprising implementation of the PROLOG SOURCE CODE set forth in the PROLOG SOURCE CODE APPENDIX.

11. The computerized system of claim 6 further comprising means for displaying, accepting and retrievably storing the test item model.

12. A computerized system for generating test item variants comprising:

- a. means for creating, editing and storing variabilized and non-variabilized test items;
- b. means for selectively variabilizing test item elements;
- c. means for defining test item element variables;
- d. means for simultaneously solving variabilized test item element constraints;
- e. means for displaying and storing accepted test items.

13. A computerized method for generating test item variants from test item models comprising:

- a. retrievably storing test item models;
- b. selecting a test item model;
- 5 c. simultaneously solving test item model constraints and generating test item solutions;
- e. displaying, accepting and retrievably storing valid test item solutions.

14. The computerized method of claim 13 wherein the step of
10 retrievably storing test items models comprises:

- a. obtaining a test item;
- b. identifying elements of the test item to be
variabilized;
- c. variabilizing the elements to create variables;
- 15 d. defining the variables;
- e. accepting the variabilized test item with defined
variables as a test item model.

15. The computerized method of claim 13 further comprising
specifying constraints that define the relationship among he
20 variables.

16. The computerized method of claim 14 further comprising the
steps of displaying and retrievably storing the accepted test
item model.

17. The computerized method of claim 14 wherein the test item model constraints are simultaneously solved using PROLOG IV and TCA constraint language.

18. The computerized method of claim 17 wherein the Prolog simultaneous constraint solver is the PROLOG SOURCE CODE set forth in the PROLOG SOURCE CODE APPENDIX.

19. The computerized method of claim 14 wherein variables can be defined by values which are variables.

20. The computerized method of claim 15 wherein the variables are new variables for which new constraints are defined as needed.

ABSTRACT

A computerized method and system for creating test items by generating variants from a test item model, comprising the steps of creating a new test item model by identifying elements of an initial test item or test item model to be variabilized, variabilizing the elements thereby creating test item variables, indicating values the variables can assume, defining the variables, and generating test item variants utilizing a simultaneous constraint solver. The initial test item can be a pre-existing test item or test item model, a newly created test item or even a conceptual template in the mind of the test item creator. The generated test item variants are displayed to the test item creator. The test item creator can store and forward acceptable test item variants for later use as test items. Test item models can be stored for later use in generating new test item variants.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT EXAMINING OPERATION

ATTN'Y DOCKET NO.: ETS-TCA

APPLICATION OF: PETER BRITTINGHAM, MARY E. MORLEY, MARK K.
SINGLEY, MARK G. ZELMAN, KRISHNA N. JHA,
JAMES H. FIFE, ROBERT L. RARICH, IRVIN R.
KATZ, RANDY E. BENNETT

FOR: COMPUTER-BASED TEST-ITEM GENERATION AND
CLONING

DRAWINGS

(FIGS. 1-73, 73A-73E, 74-79, 80A-
80C, 81, 82A-82C, 83-97, 98A-98B,
99-105, 106A-106B, 107)

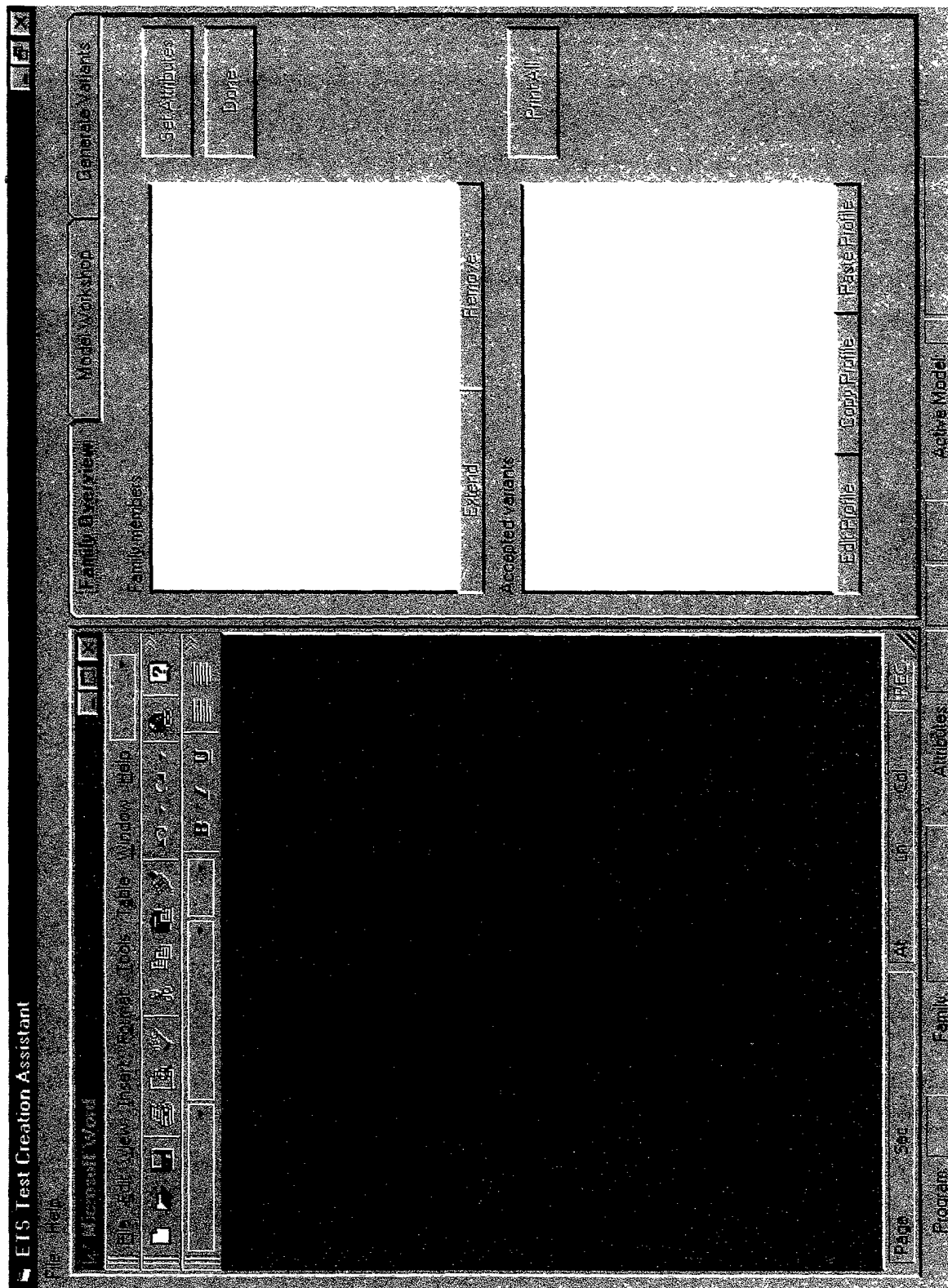


FIG. 1

ETS Test Creation Assistant

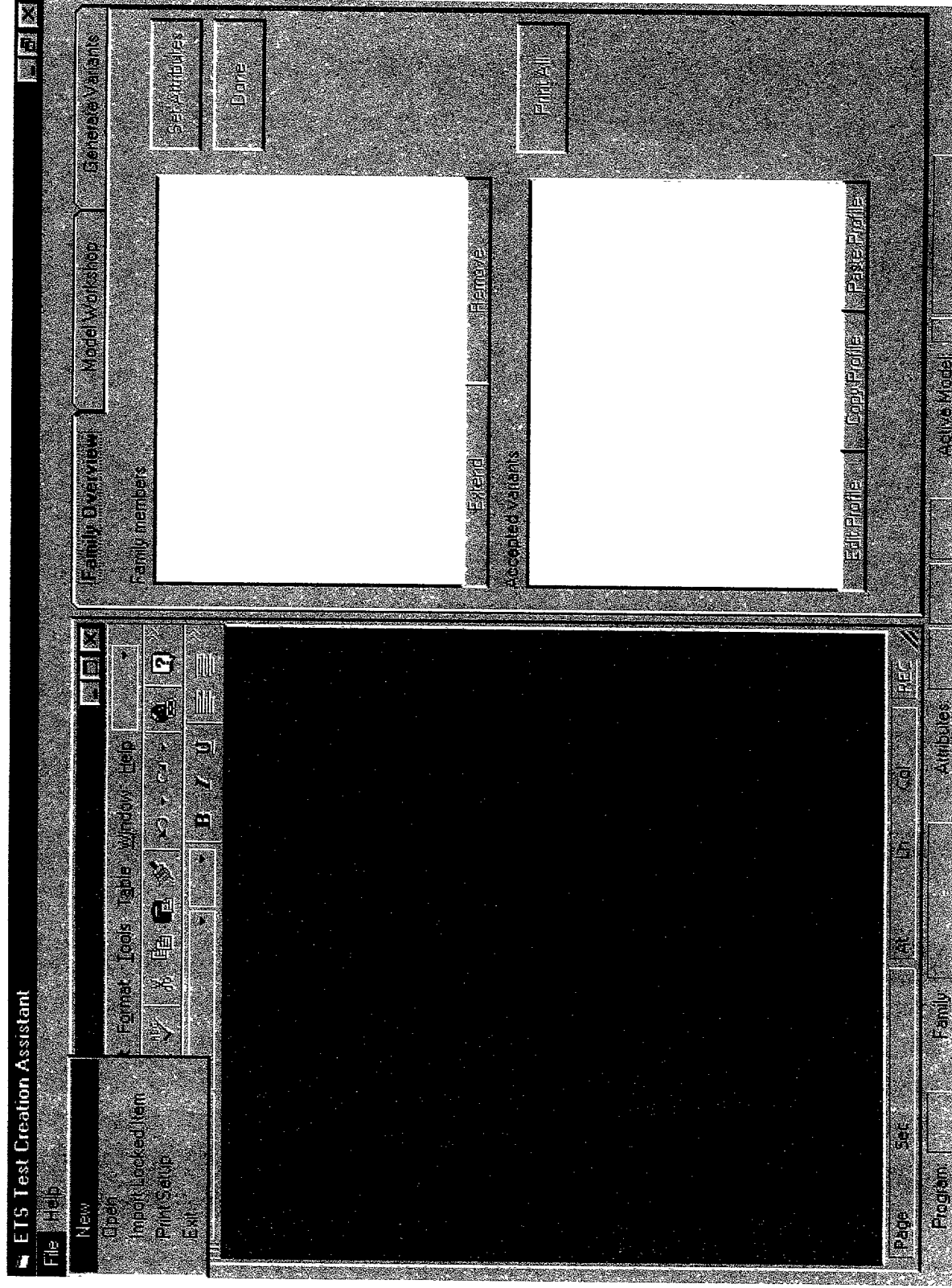


FIG. 2

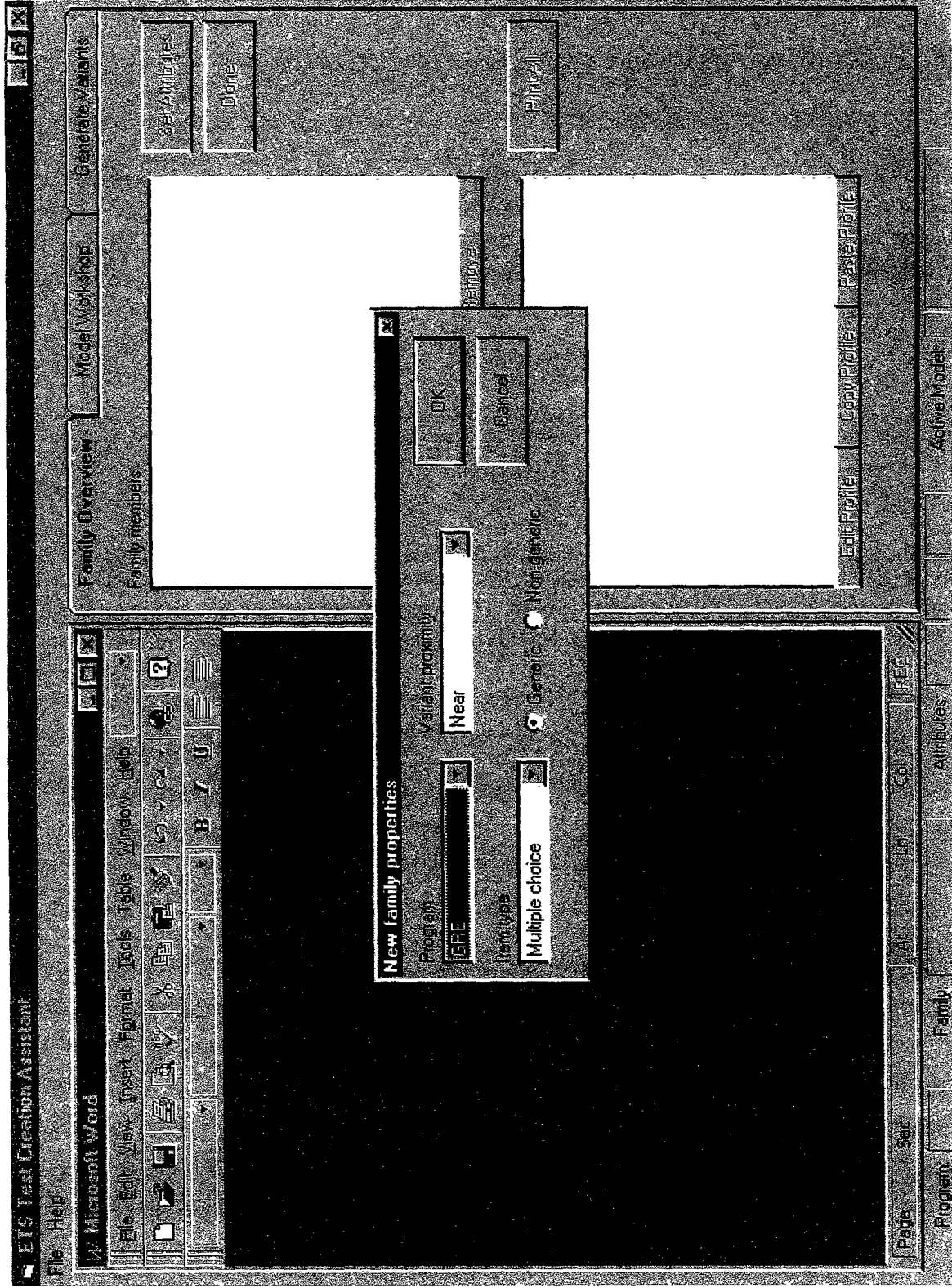


FIG. 3

Microsoft Word 6.0.2.2672
Copyright 1993 Microsoft Corporation
All rights reserved. Microsoft, the Microsoft logo, Word, and the Word logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

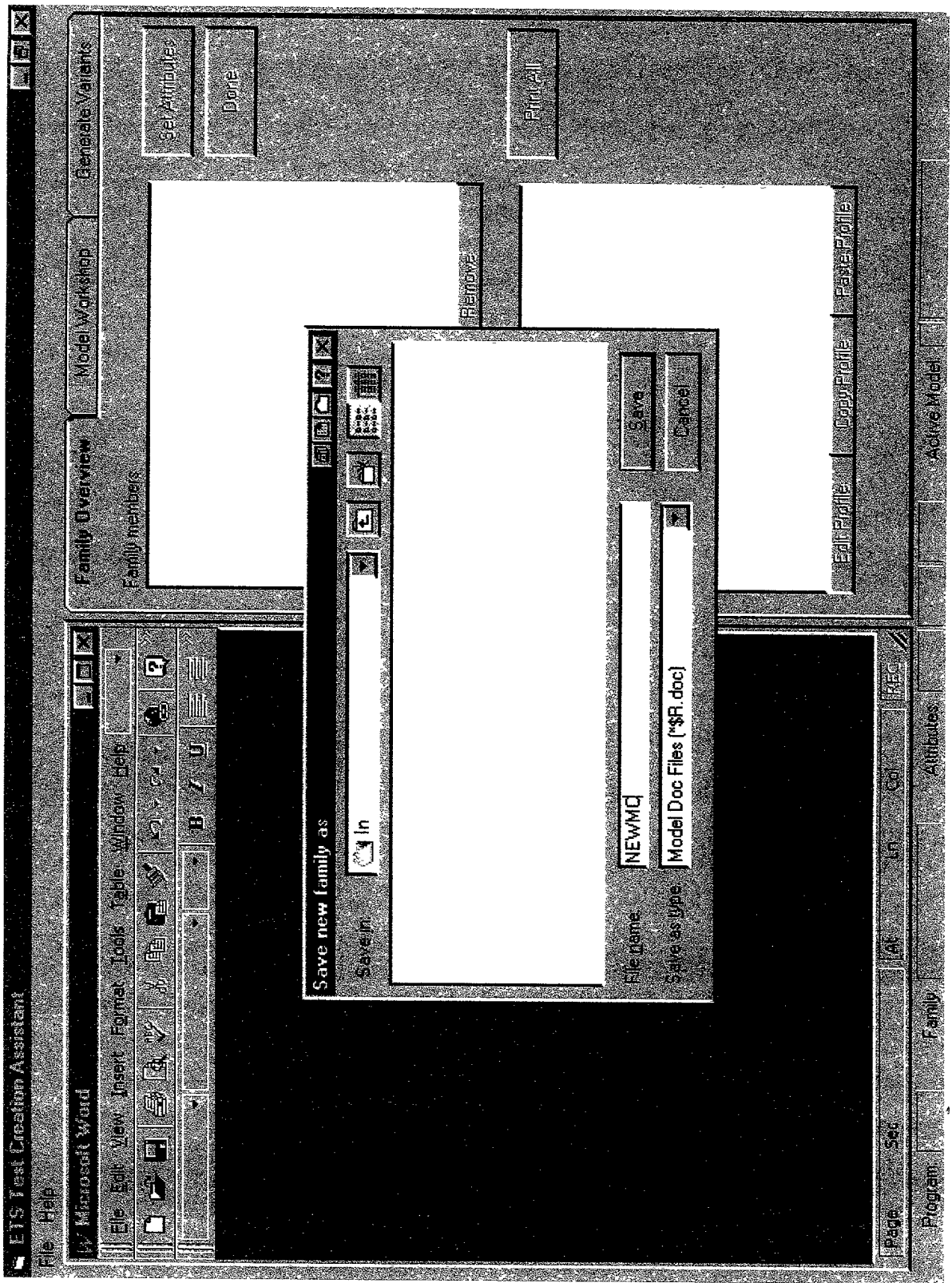


FIG. 4

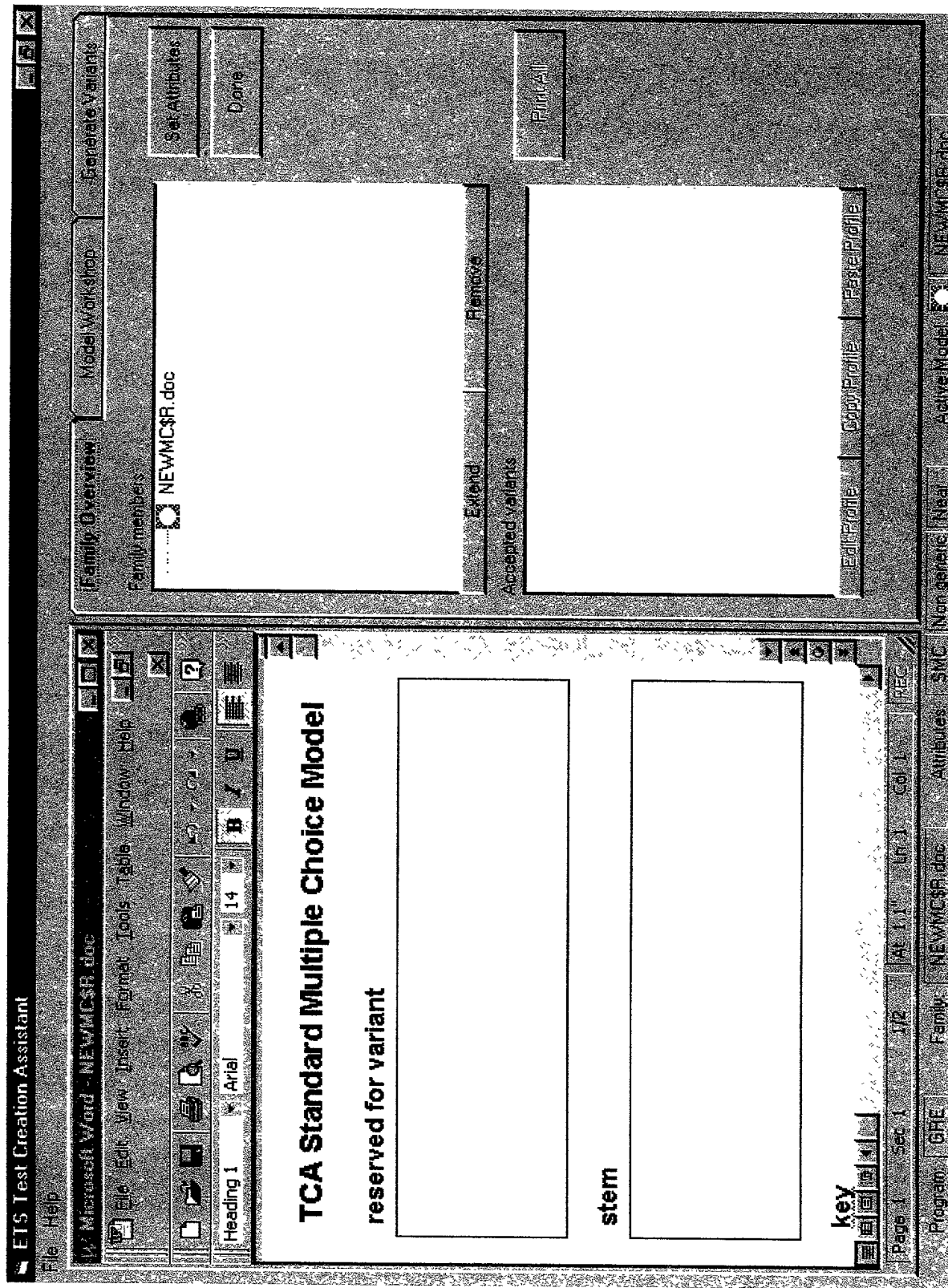


FIG. 5

TCA Standard Multiple Choice Model

reserved for variant

stem

key

Key

distractor1

Distractor1

distractor2

Distractor2

distractor3

Distractor3

distractor4

Distractor4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch
Pad
Area

FIG. 6

Microsoft Word - NEWMC\$R.doc

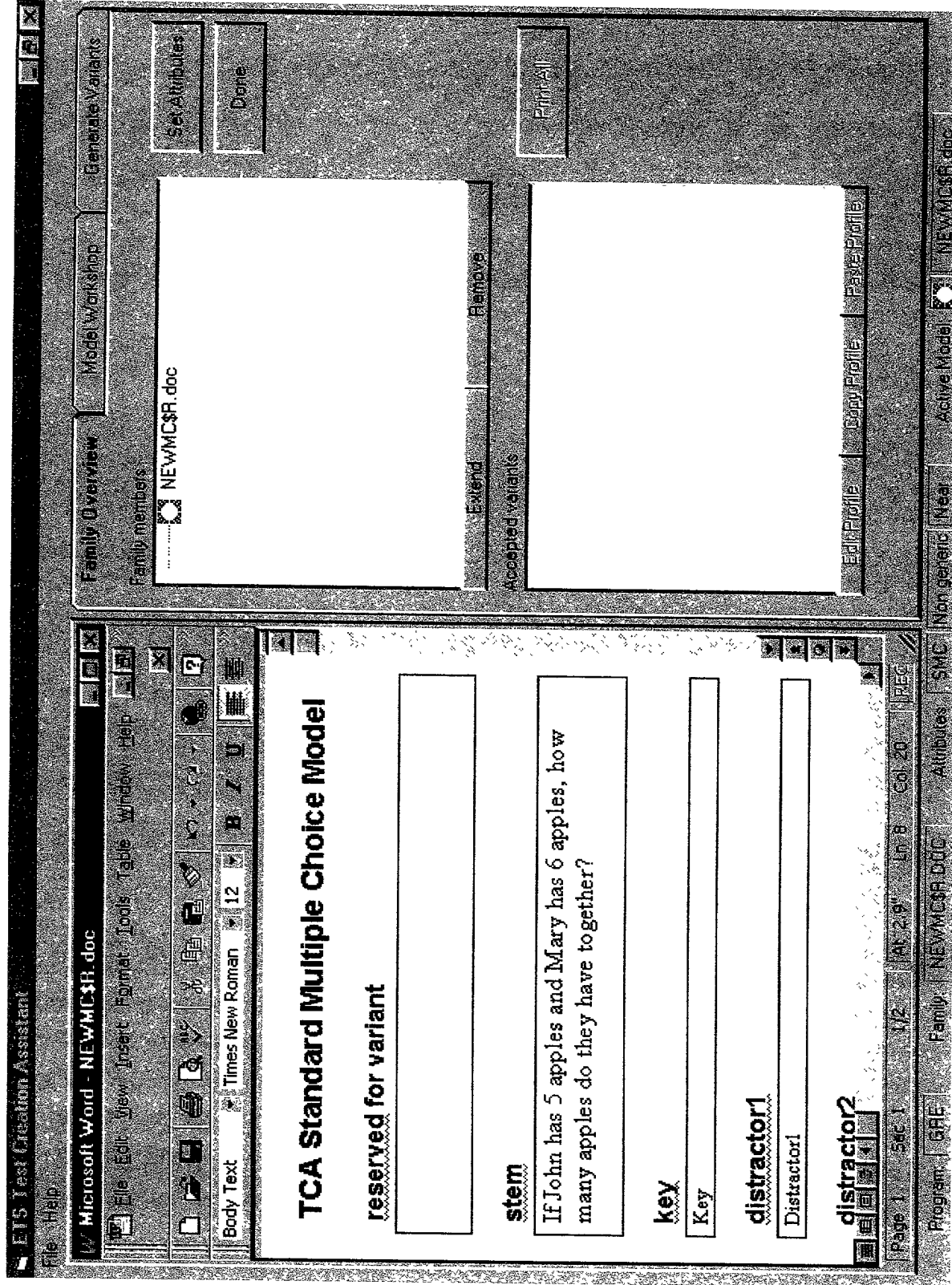


FIG. 7

Microsoft Word - NEWMC\$R.doc

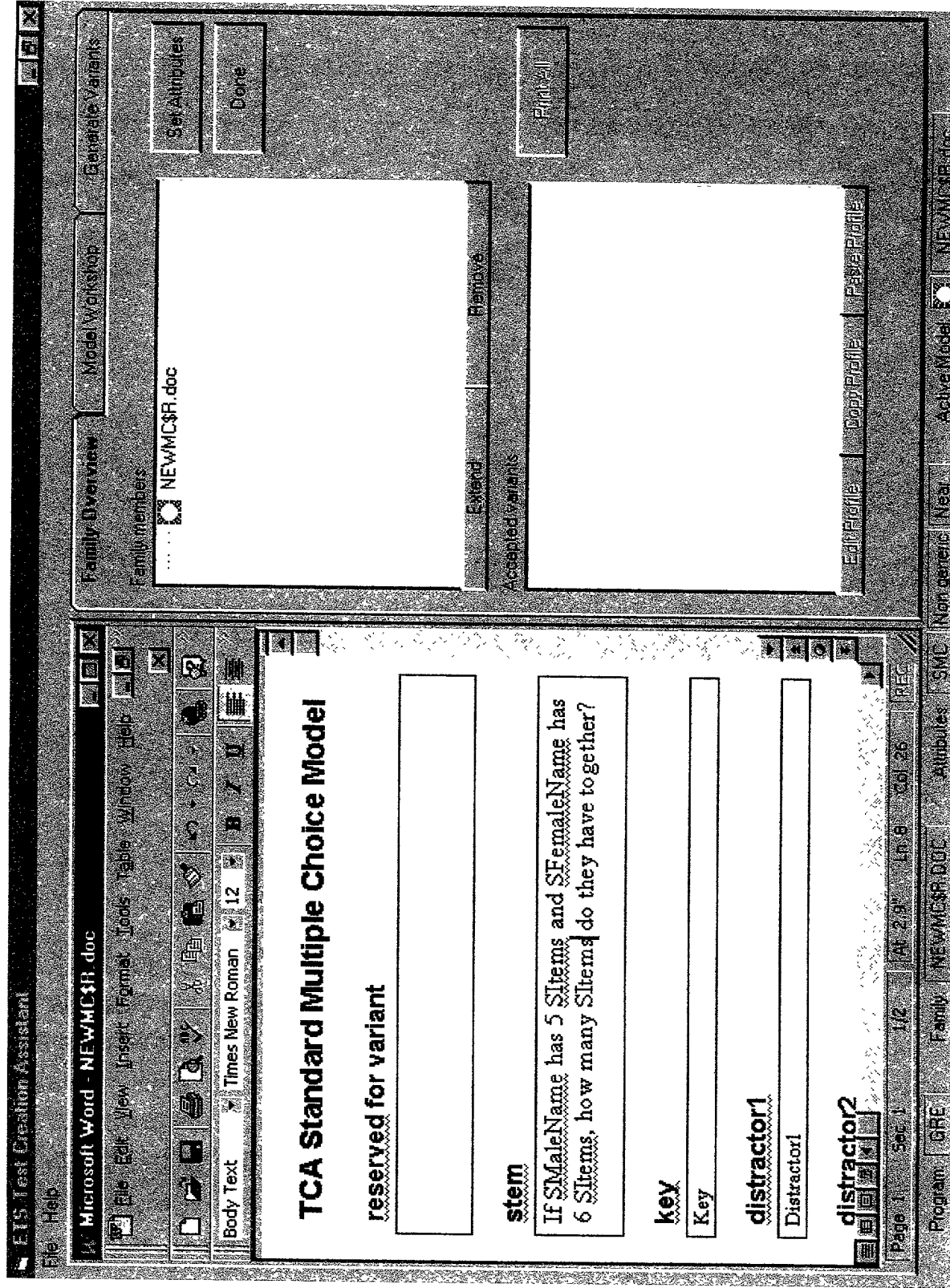


FIG. 8

QUESTIONS: How many items do they have together?

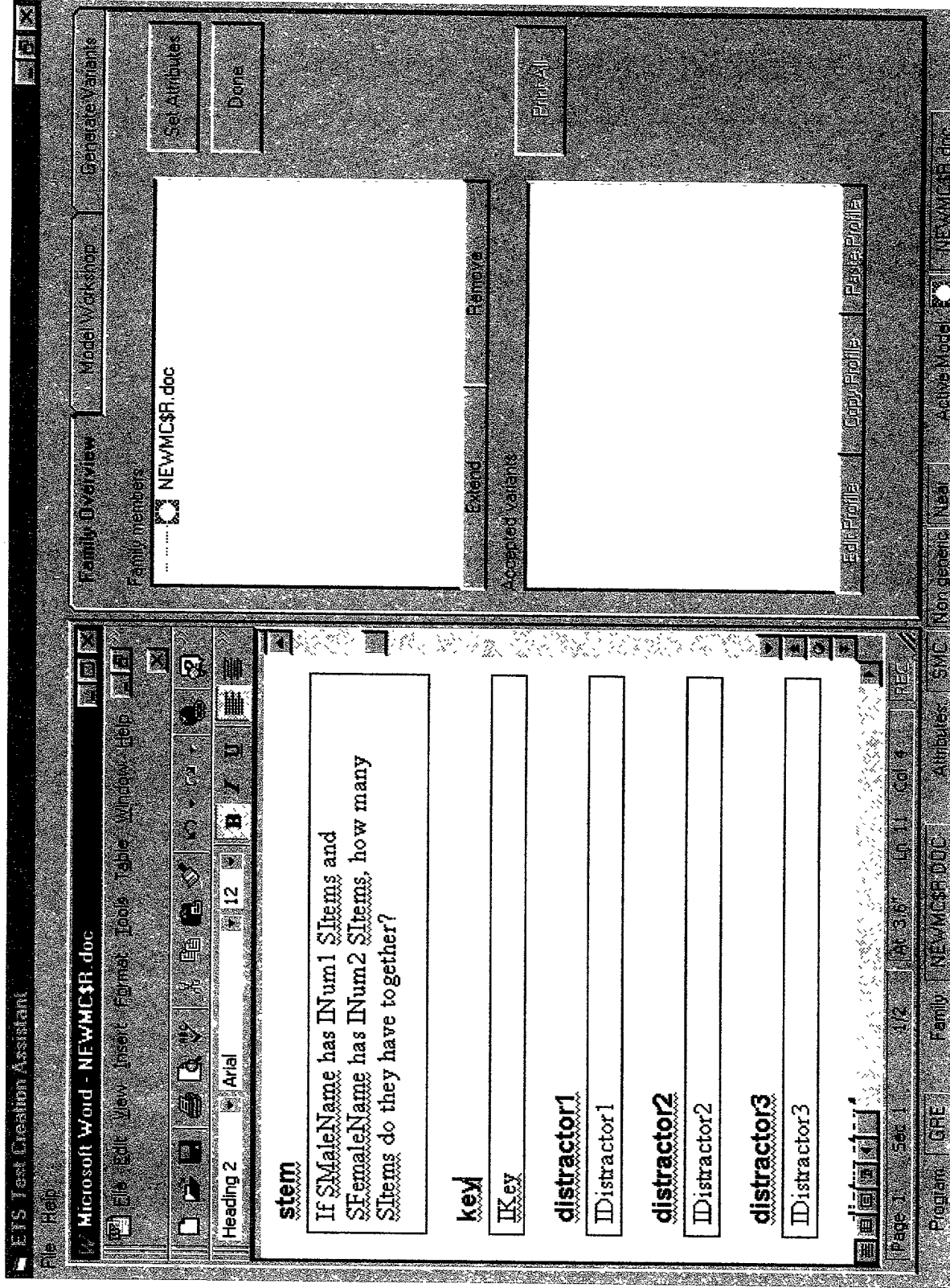


FIG. 9

Microsoft Word - NEWMC\$R.doc
File Edit View Insert Format Tools Table Window Help
B I U L A C O S P
stem
key
distractor1
distractor1
distractor2
distractor2
distractor3
distractor3
NEWMC\$R.doc

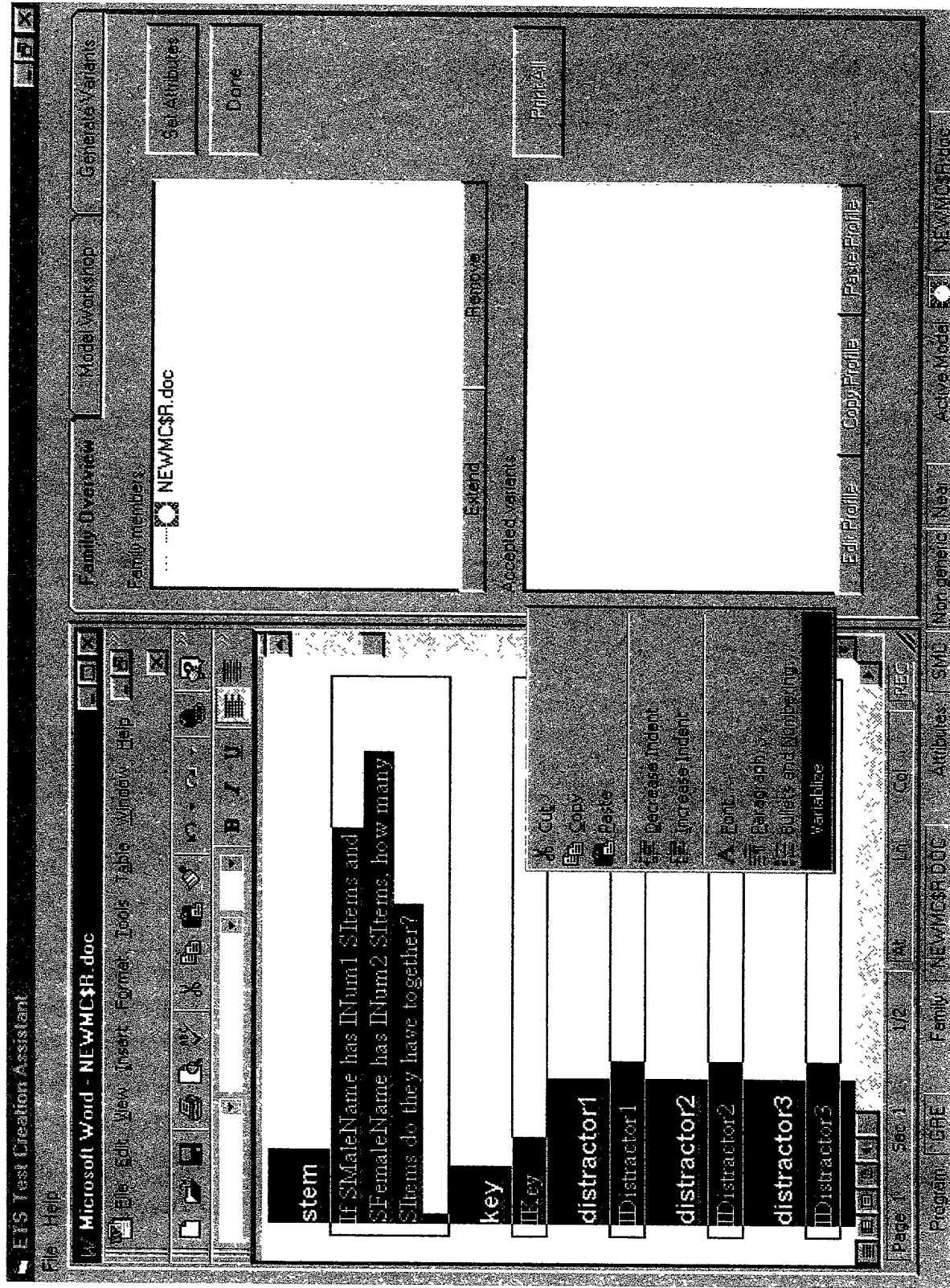


FIG. 10

FIG. 11 is a screenshot of the Microsoft Word 6.0 application window showing a document titled "NEWMC\$R.doc". The document content is a list of items: "stem", "key", "distractor1", "distractor2", "distractor3", and "distractor4". The "stem" item is highlighted. A "Print" button is visible in the top right corner. The status bar at the bottom indicates "Page 1 of 1" and "100 words".

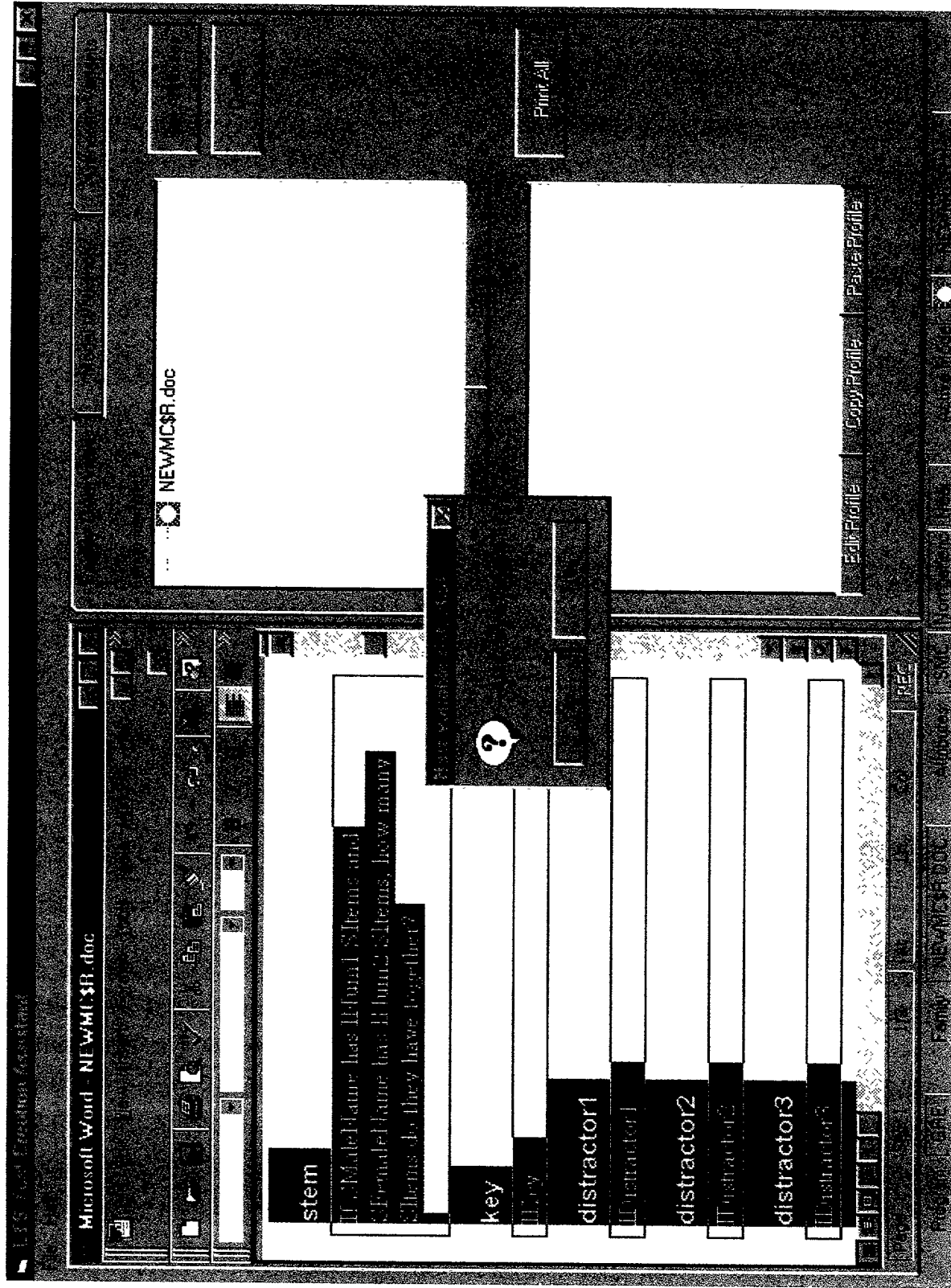


FIG. 11

FIG. 12 is a screenshot of the NEWMCSPR.doc file in the Microsoft Word 6.0 program. The screenshot shows the "New variable detected" dialog box, which is a small window with a question mark icon and the text "New variable detected". The dialog box is open over the main text area of the document. The main text area contains the following text: "stem", "All Maleblaine has 11 him 1 3 items and", "All Maleblaine has 11 him 2 3 items, how many", "items do they have together?", "key", "key", "distractor1", "distractor1", "distractor2", "distractor2", "distractor3", "distractor3". The dialog box is positioned over the text "key". The "New variable detected" dialog box has a "Yes" button and a "No" button. The "Yes" button is highlighted. The "No" button is not highlighted. The "New variable detected" dialog box is a small window with a question mark icon and the text "New variable detected". The dialog box is open over the main text area of the document. The main text area contains the following text: "stem", "All Maleblaine has 11 him 1 3 items and", "All Maleblaine has 11 him 2 3 items, how many", "items do they have together?", "key", "key", "distractor1", "distractor1", "distractor2", "distractor2", "distractor3", "distractor3". The dialog box is positioned over the text "key". The "New variable detected" dialog box has a "Yes" button and a "No" button. The "Yes" button is highlighted. The "No" button is not highlighted.

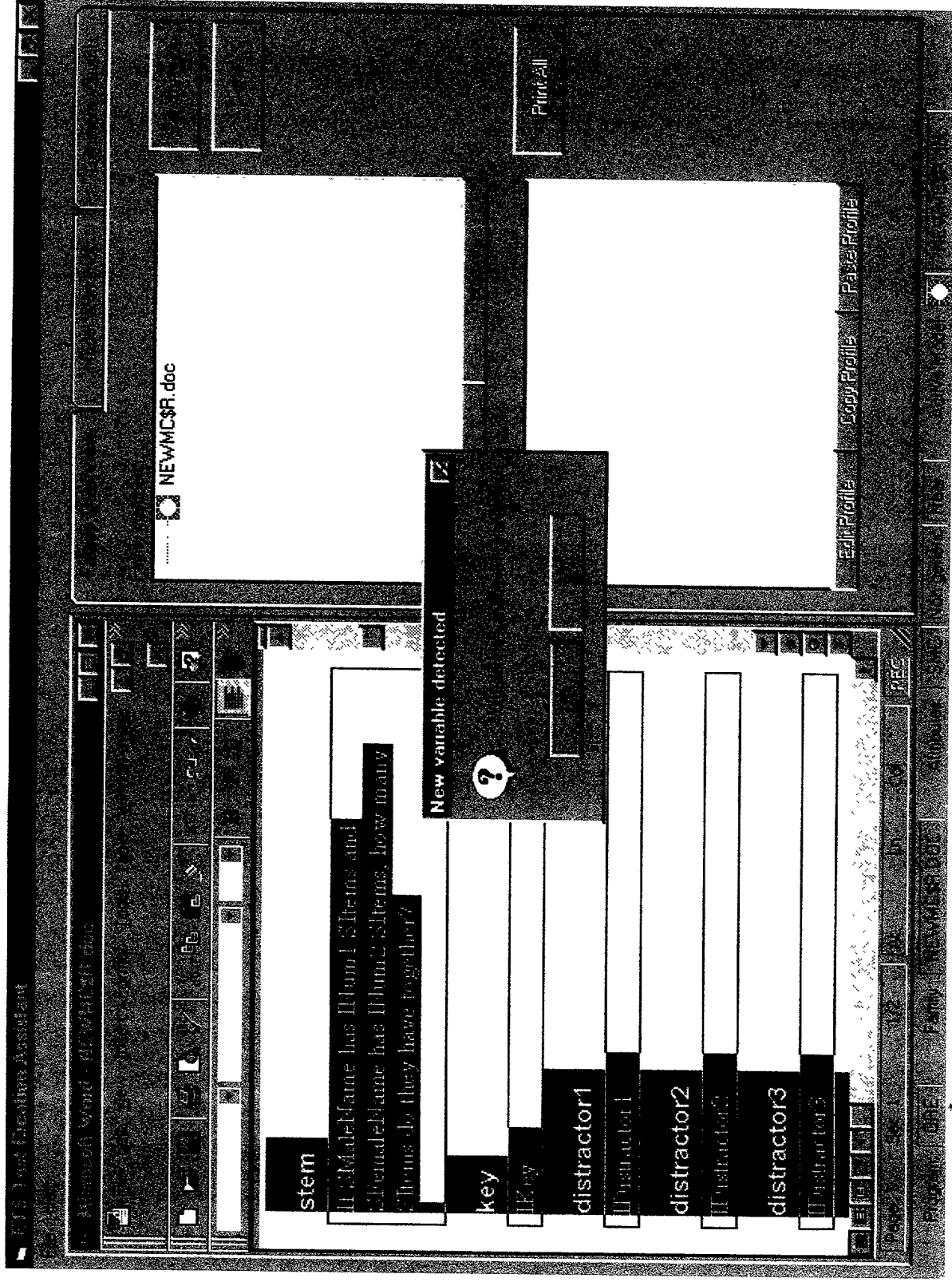


FIG. 12

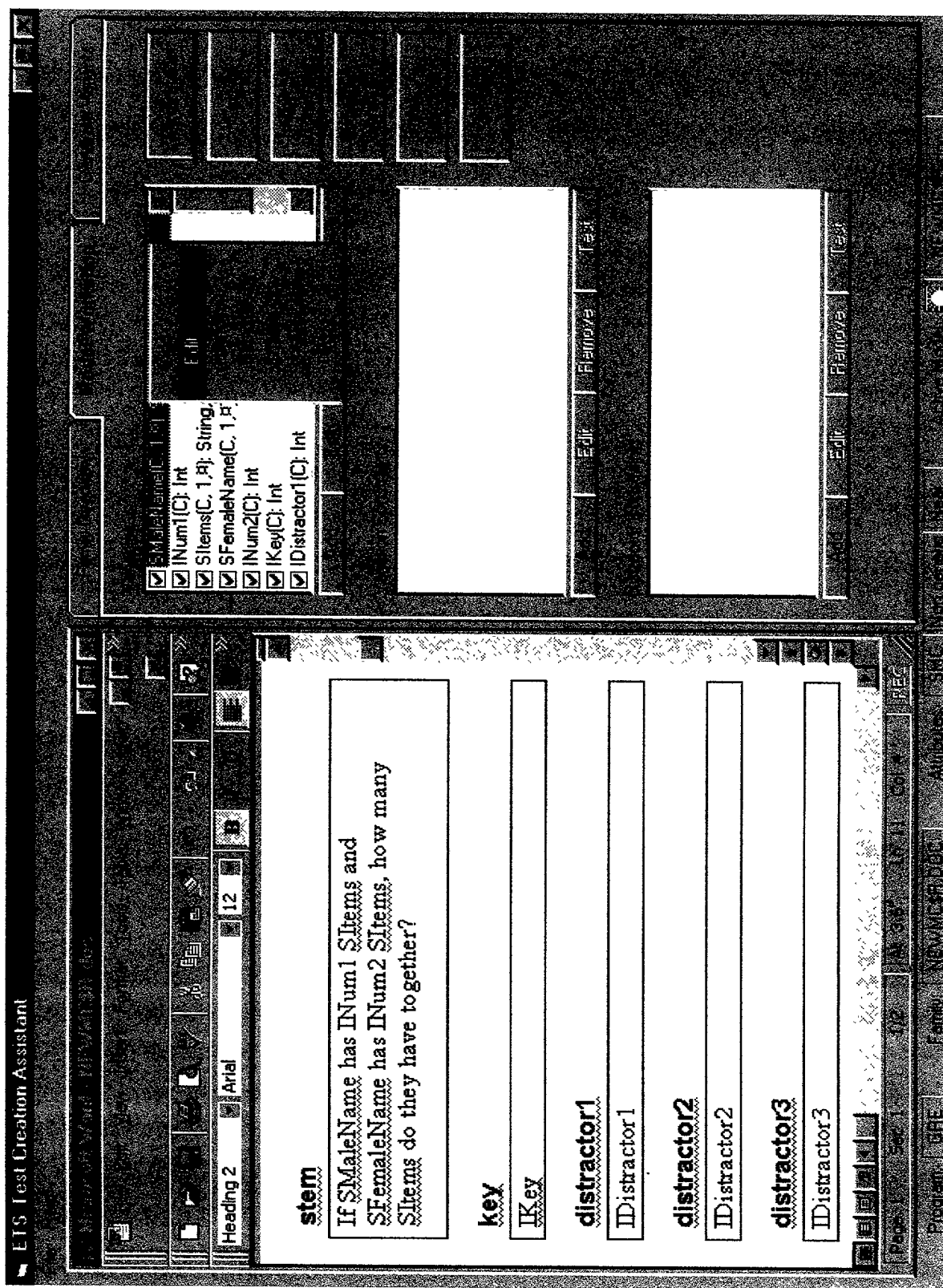


FIG. 14

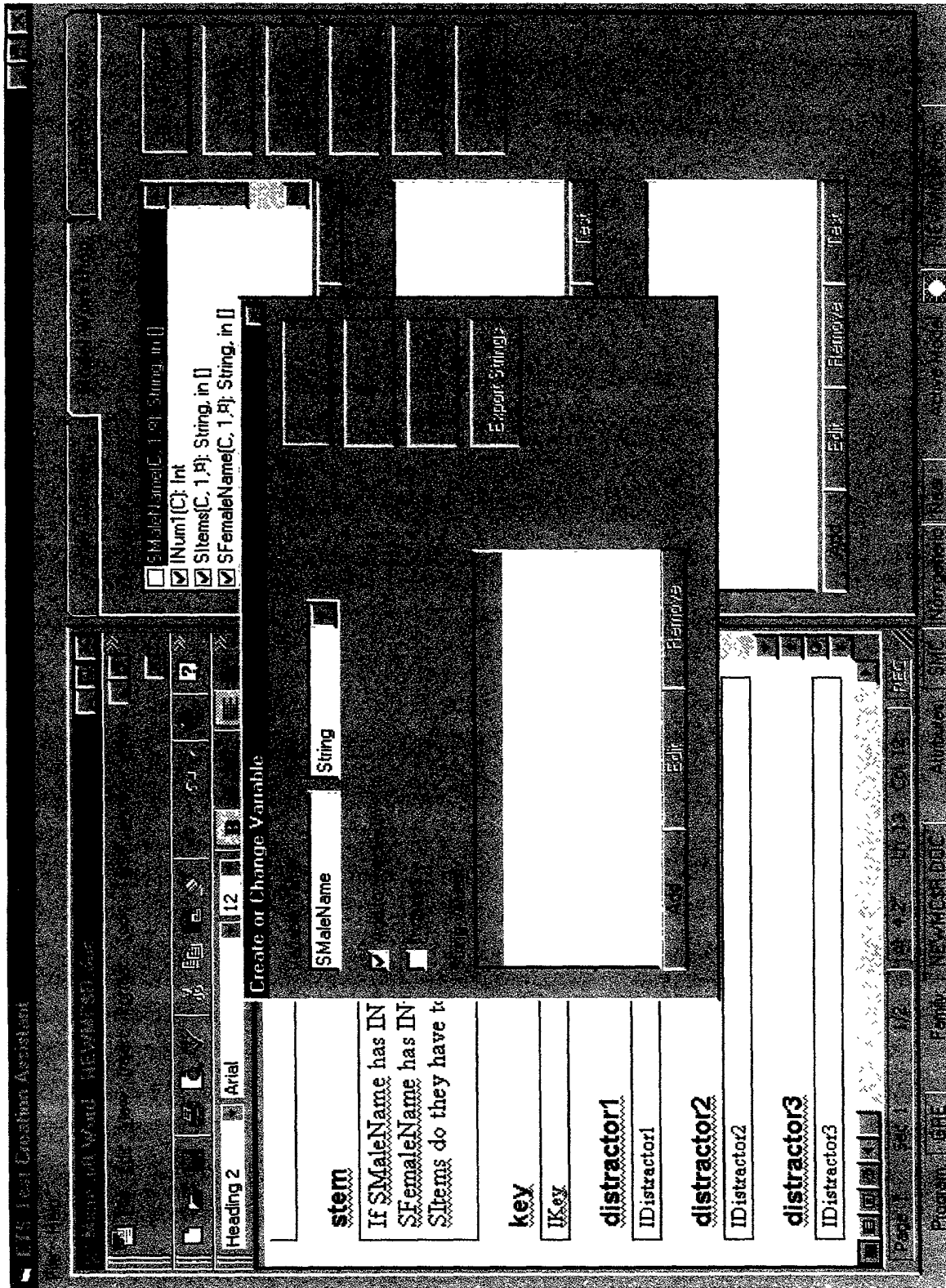
[illegible]

FIG. 16

FIG. 17 is a screenshot of the EDB User Creation Assistant window. The window displays a list of fields for user creation, including SMaleName, key, distractor1, distractor2, and distractor3. The SMaleName field is highlighted, and a dialog box is open for editing the string 'John'.

EDB User Creation Assistant

Heading 2 Arial 12

SMaleName String

☒ SMaleName has IN
☒ SFemaleName has IN
☐ SItems do they have to

key
Key

distractor1
IDistractor1

distractor2
IDistractor2

distractor3
IDistractor3

Editing string SMaleName

John

Export Strings

Page 1 of 1

Program GRE Family NEURONSHIP CODE Attributes SMC Not Selected Active Mode

FIG. 17

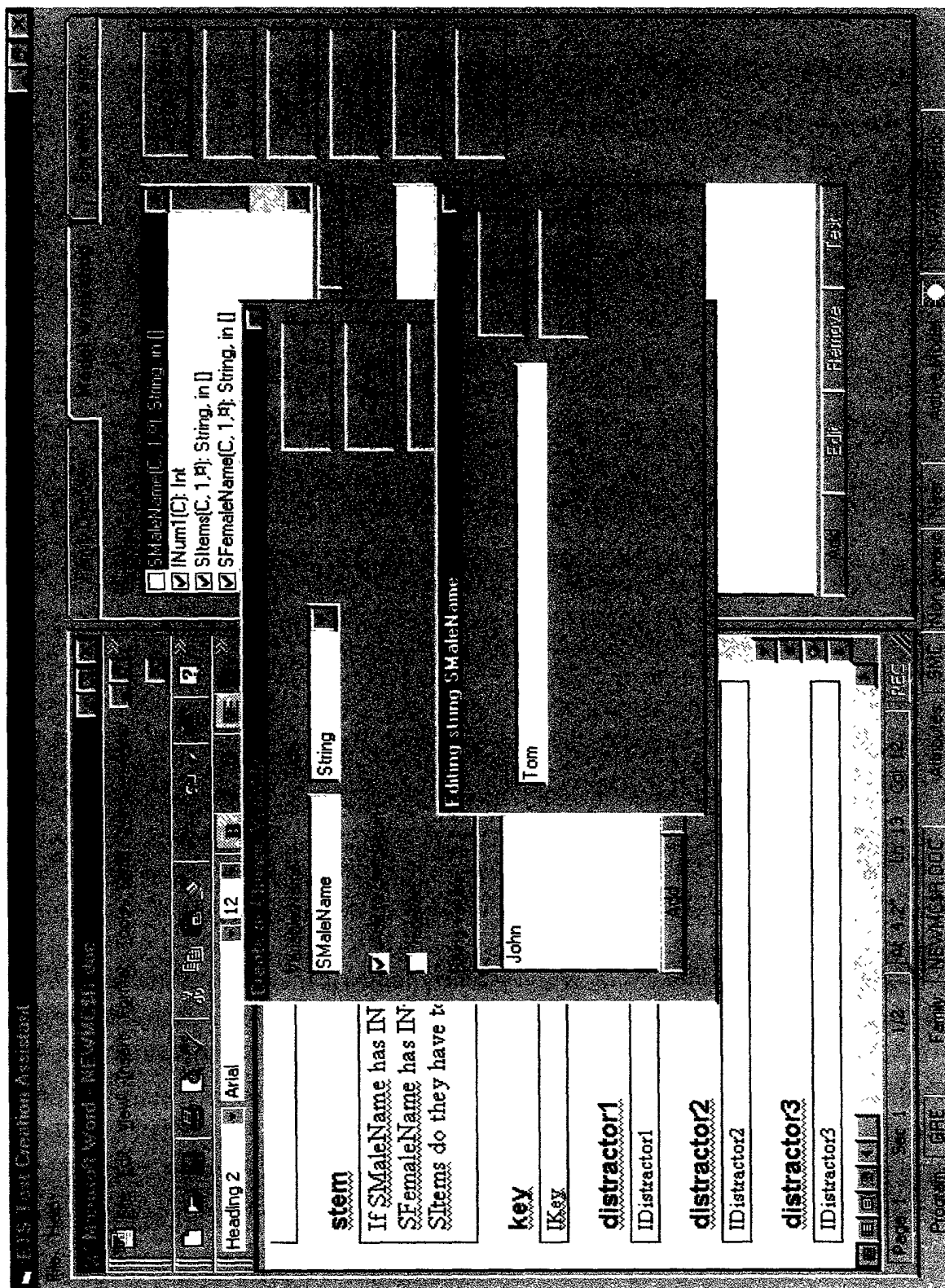


FIG. 18

Copyright 1999 by Microsoft Corporation. All rights reserved. Microsoft, the Microsoft Dynamics logo, and the Microsoft Dynamics logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

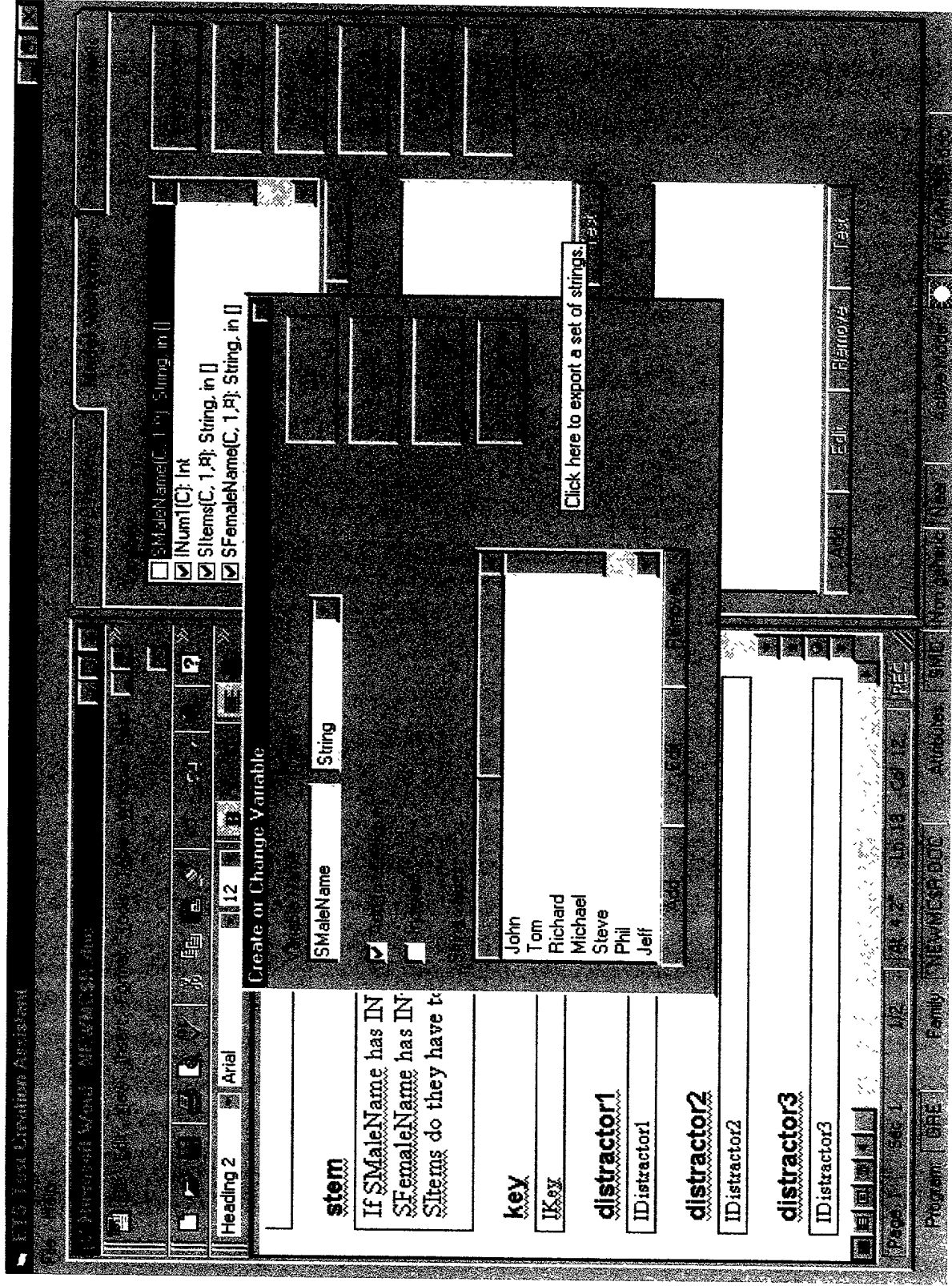


FIG. 19

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

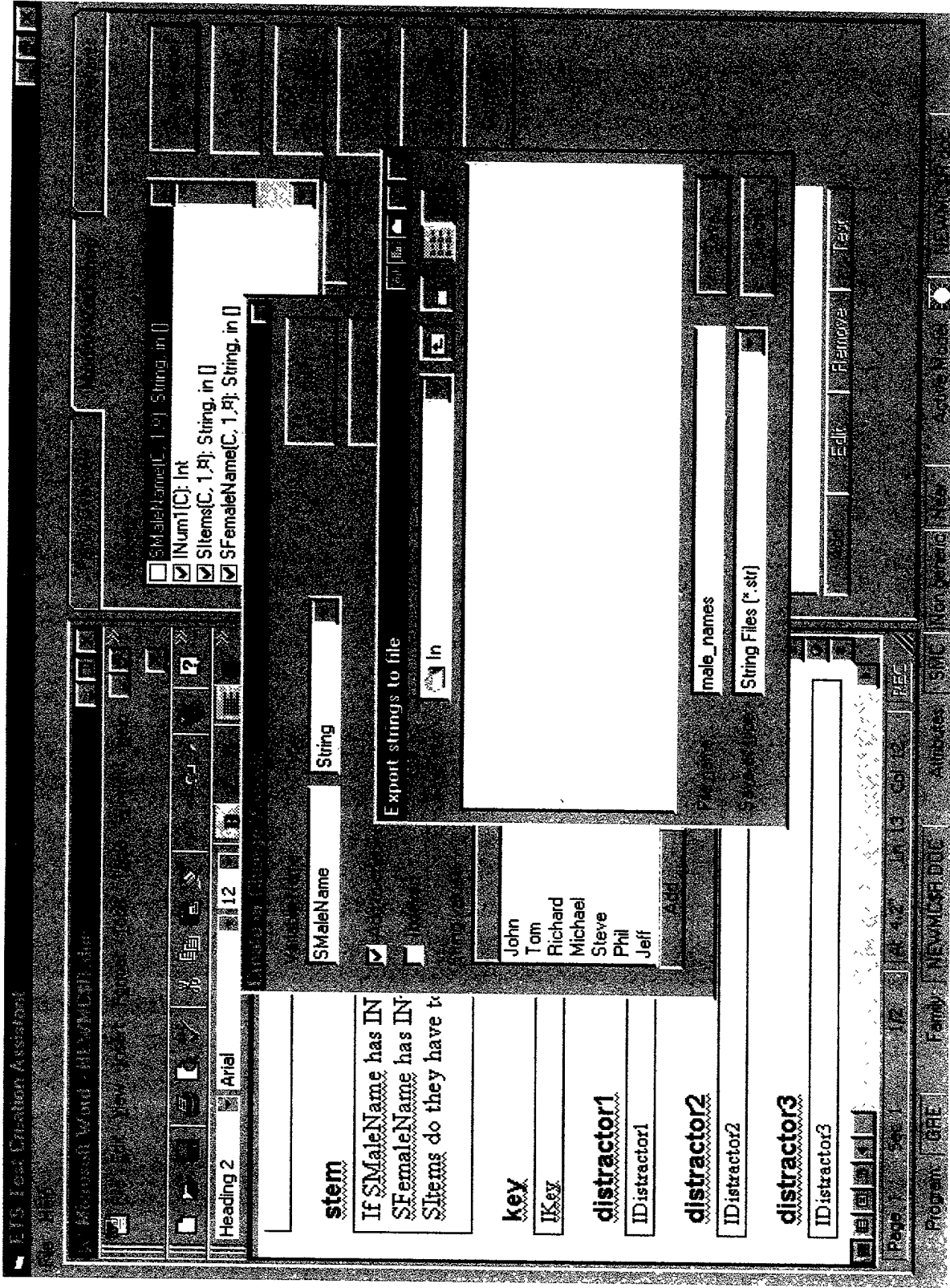


FIG. 20

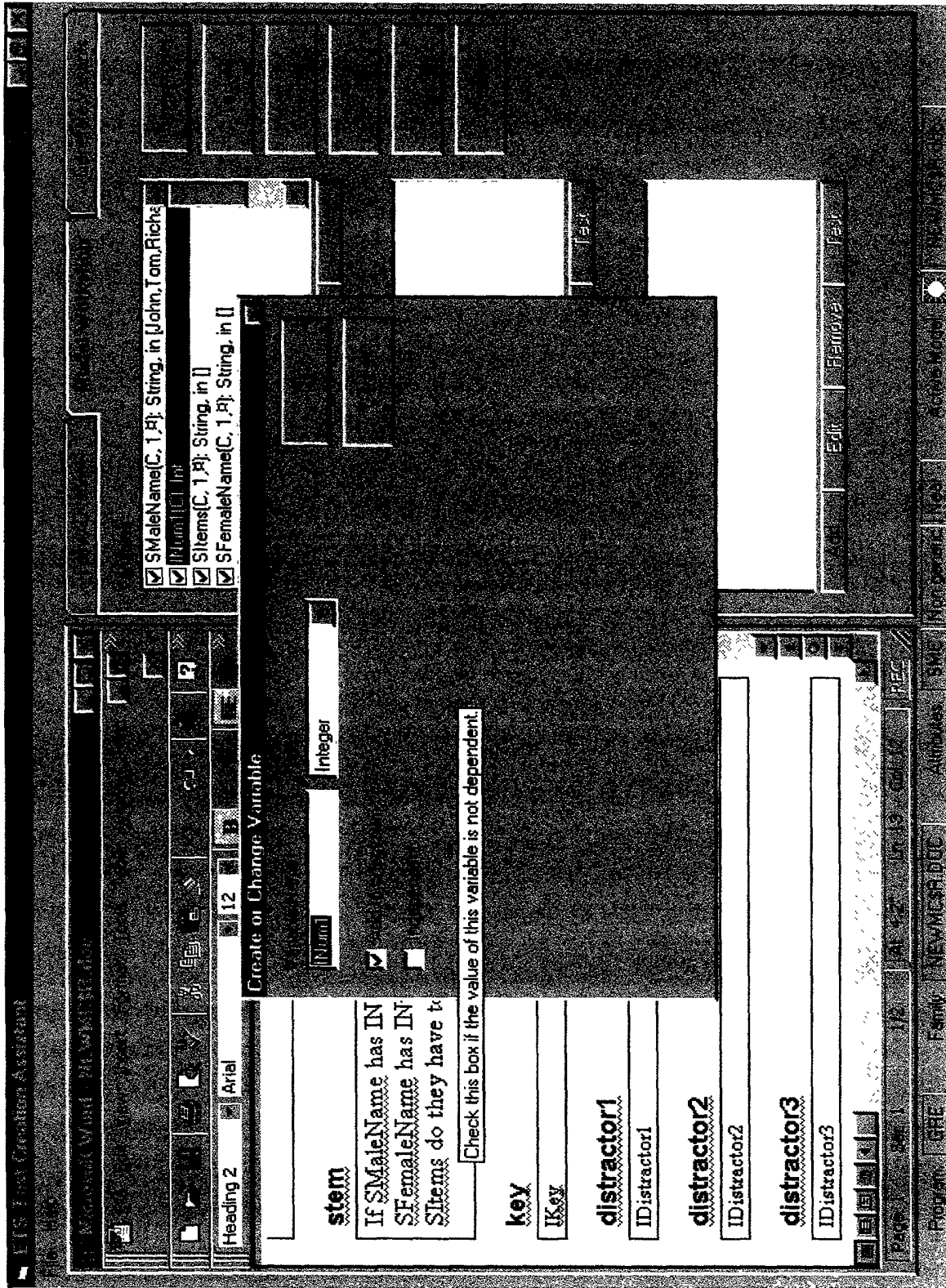


FIG. 21



FIG. 22

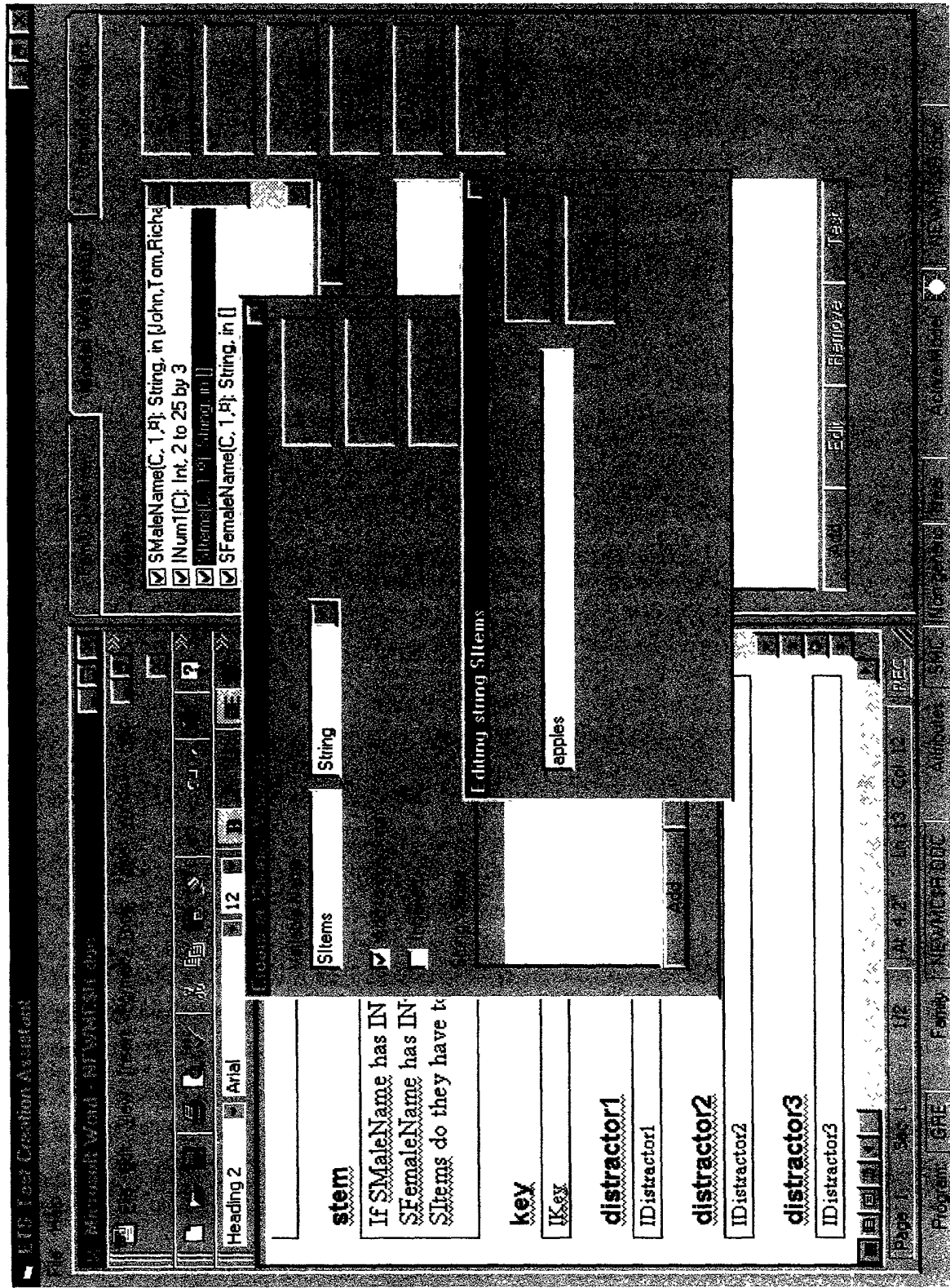


FIG. 24

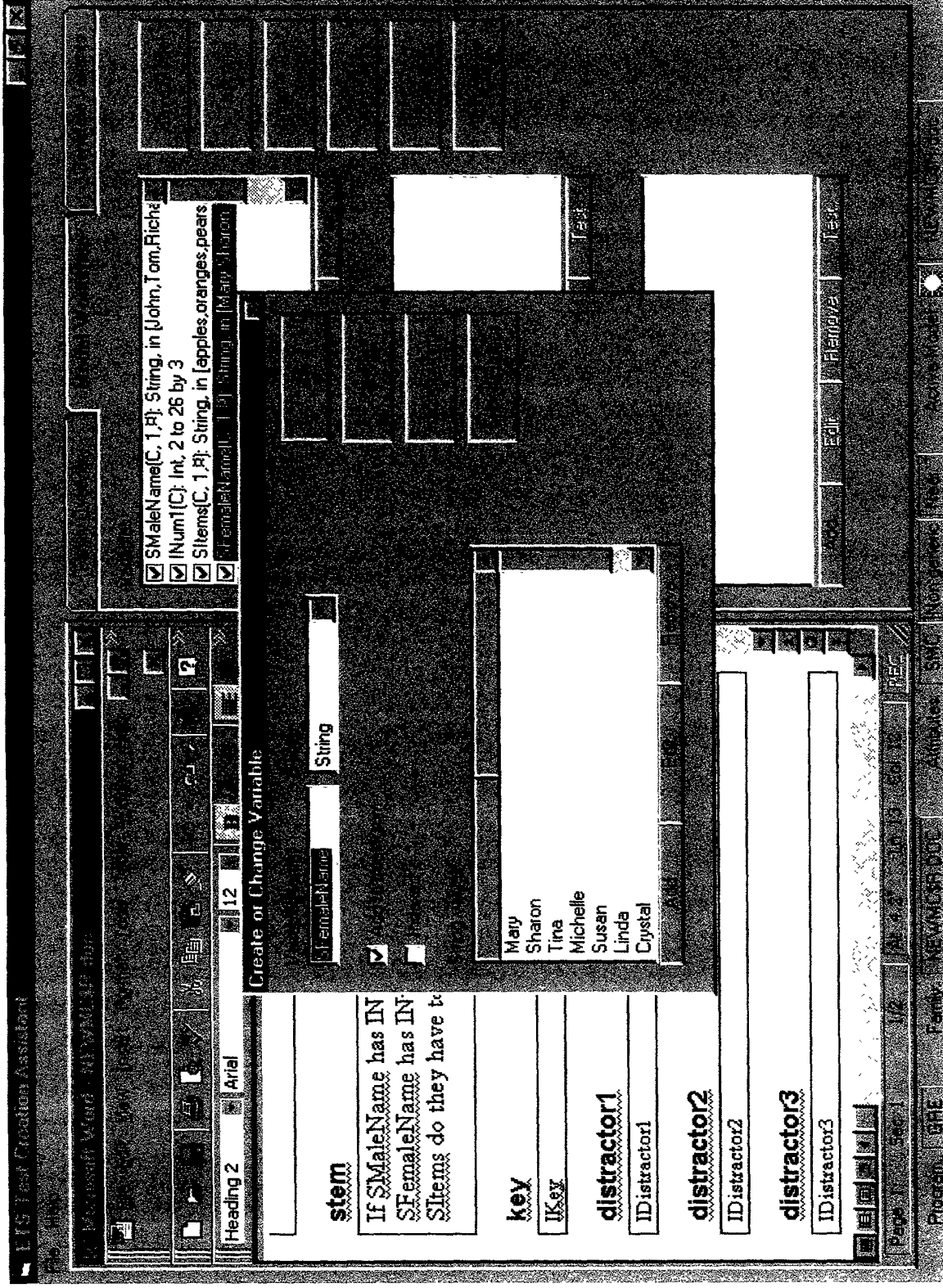


FIG. 26



FIG. 28

Microsoft Word - NEWMC\$R.doc

EIS Test Creation Assistant

File Help

Microsoft Word - NEWMC\$R.doc

File Edit View Insert Format Tools Table Window Help

Heading 2 Arial 12 B I U

stem

If SMaleName has INum1 SItems and SFemaleName has INum2 SItems, how many SItems do they have together?

key

IKKey

distractor1

IDistractor1

distractor2

IDistractor2

distractor3

IDistractor3

Family Overview

Model Workshop

Generate Variants

Variables

☒ SFemaleName(C: 1-R) String, in [May, Sharon, ...]

☒ INum2(C): Int, 2 to 27 by 5

☒ IKKey(C): Int

☒ IDistractor1(C): Int

☒ IDistractor2(C): Int

☒ IDistractor3(C): Int

☒ IDistractor4(C): Int

Add

Edit

Remove

Test

Variation Constraints

Add

Edit

Remove

Test

Save Model

Test All

Import Constraints

Export Constraints

Print Constraints

Comments

Click here to add a variation constraint.

Distractor Constraints

Add

Edit

Remove

Test

Program: GRE

Family: NEWMC\$R.DOC

Attributes: SMC

Non-generic

Near

Active Model: NEWMC\$R.doc

FIG. 29

Microsoft Word - NEWMC\$R.doc

ETIS Test Creation Assistant

File Help

Microsoft Word - NEWMC\$R.doc

File Edit View Insert Format Tools Table Window Help

Create or Change Constraints

Constraint

OK Cancel

Operators Variables Functions

+ - * / % ^
\ | () ([1,2])
= /> < >= <=
if then else less than else self

Comment

Family Overview Model Workshop Generate Variants

Variables

☒ SFemaleName(C, 1, 9); String, in [Mary, Sharon, ...]
☒ INum2(C); Int, 2 to 27 by 5
☒ IDistractor1(C); Int
☒ IDistractor2(C); Int
☒ IDistractor3(C); Int
☒ IDistractor4(C); Int

Add Edit Remove Test

Variation Constraints

Add Edit Remove Test

Distractor Constraints

Add Edit Remove Test

Save Model
Test All
Import Constraints
Export Constraints
Print Constraints
Comments

Page 1 Sec 1 1/2 At 4.2" Ln 13 Col 12 REC

Program: GRE Family: NEWMC\$R.DOC Attributes: SMC Non generic Near Active Model: NEWMC\$R.doc

FIG. 30

Microsoft Word - NEWMC\$R.doc

File Edit View Insert Format Tools Table Window Help

Create or Change Constraints

Constraint

random() ceiling() cubert() event() eq_vars() floor() god() isbound()

Insert

clean (0)

Comment

OK Cancel

Page 1 Sec 1 1/2 At 4 1/2" Ln 13 Col 12 REC

FIG. 32

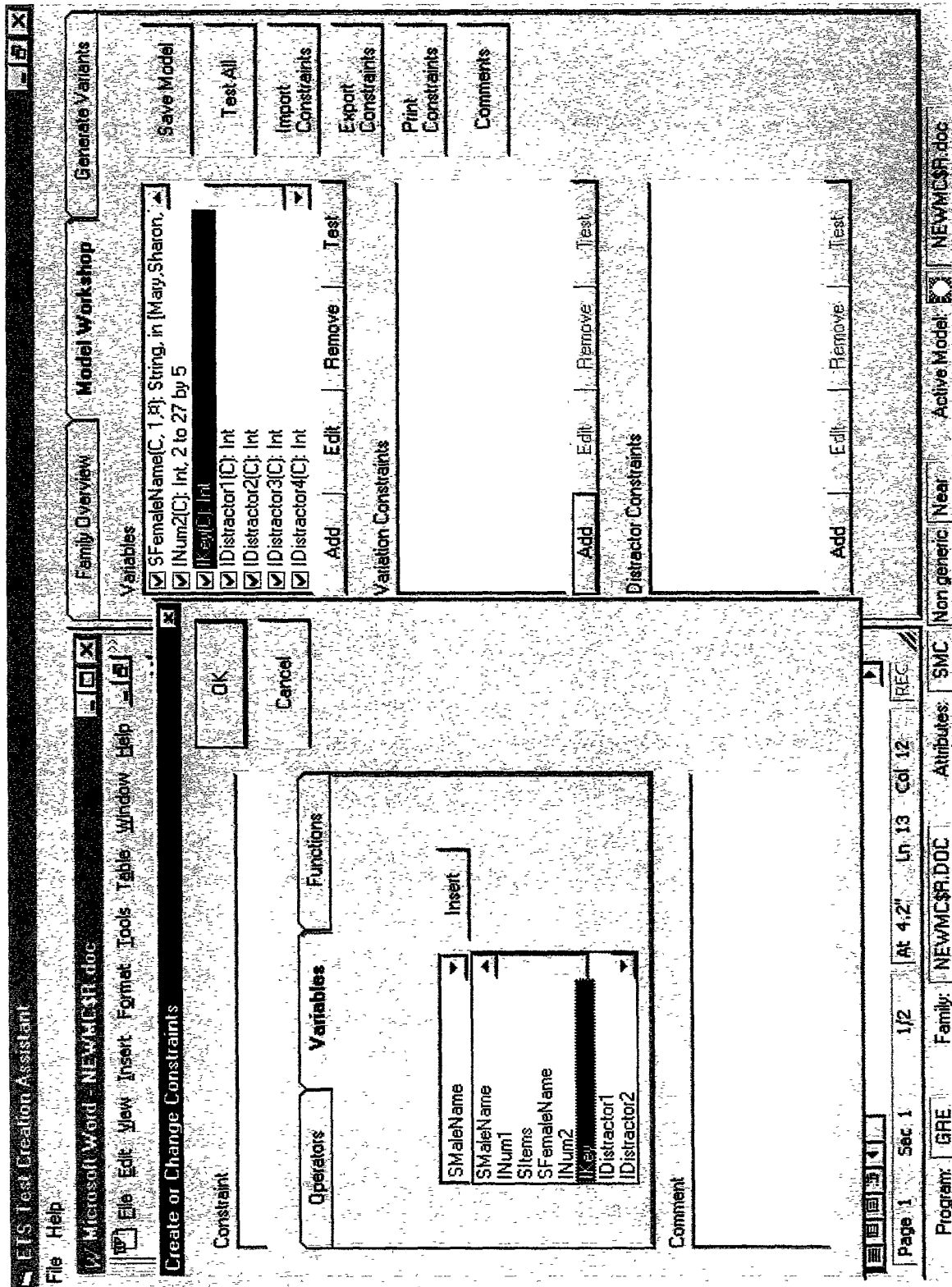


FIG. 33

FIG. 34

Microsoft Word - NEWMCSR.doc

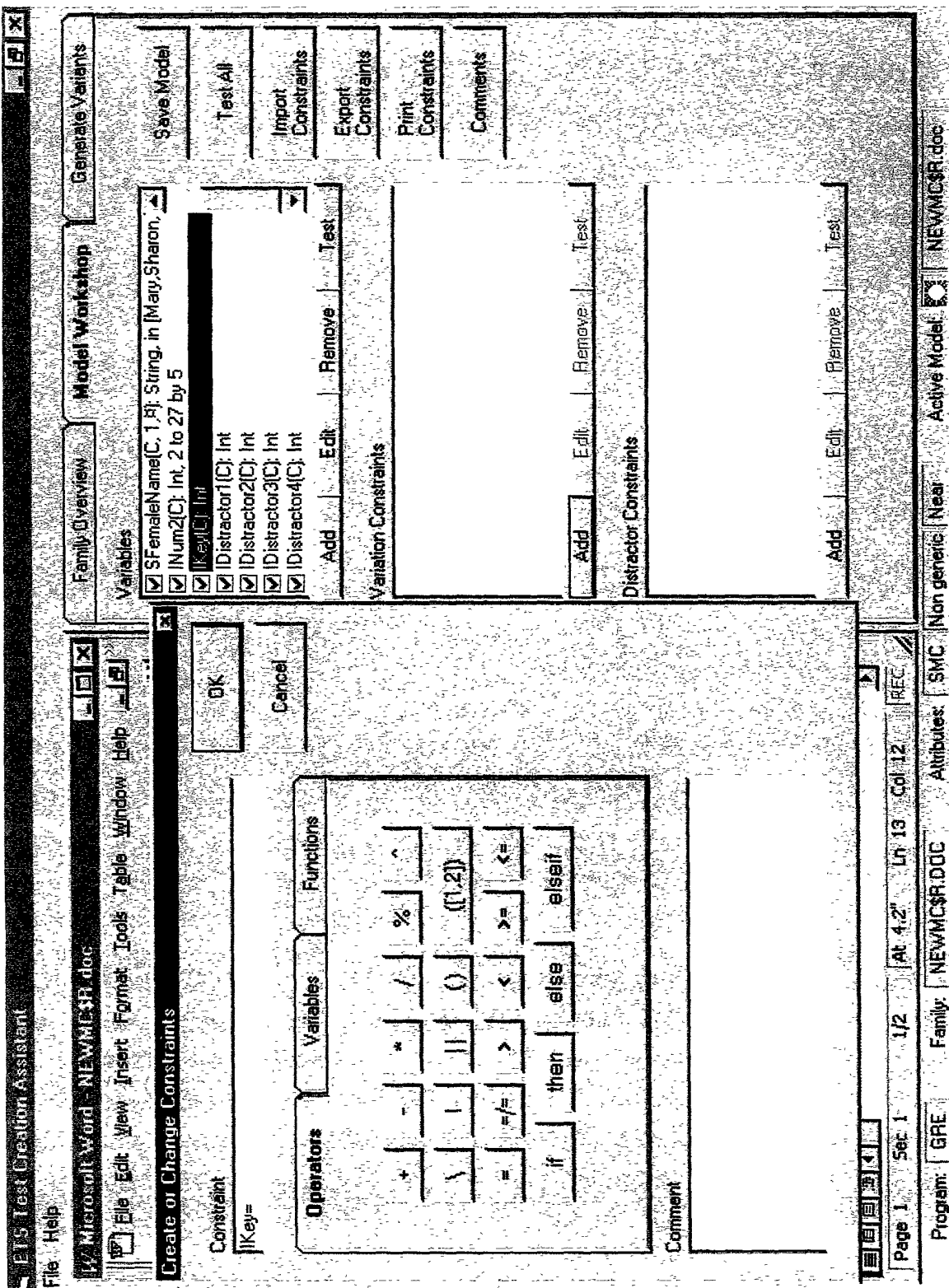


FIG. 35

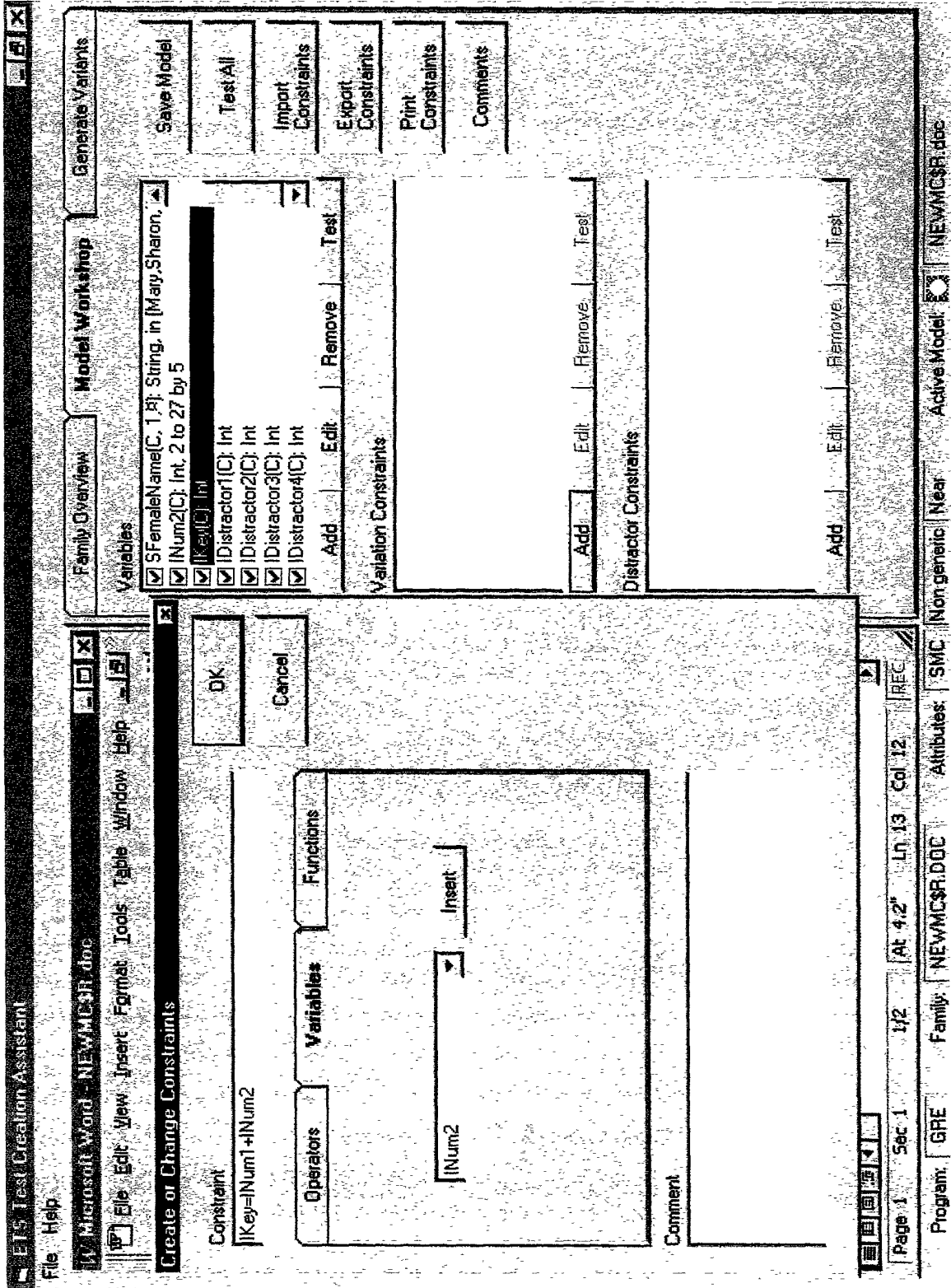


FIG. 36

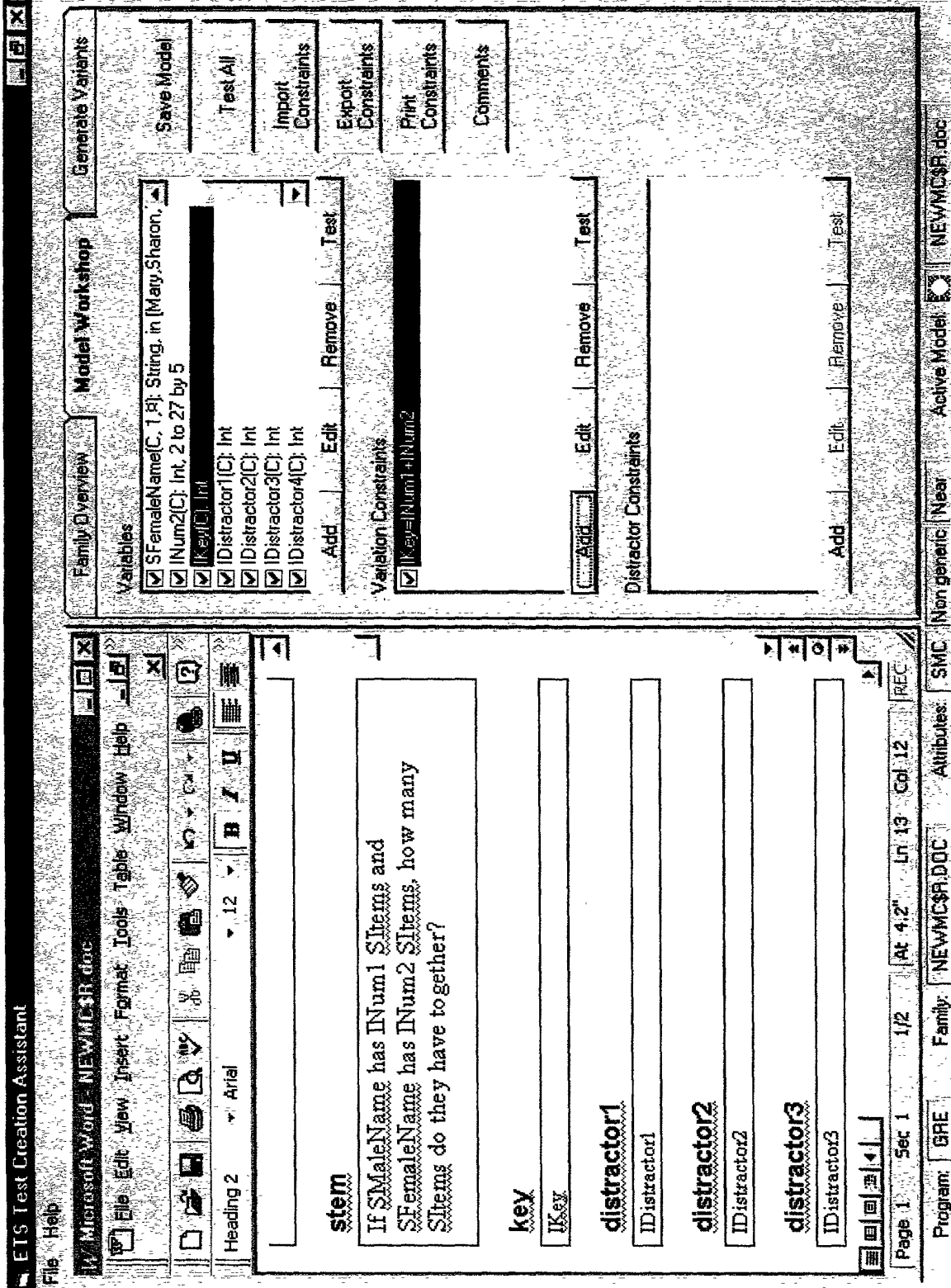


FIG. 37

FIG. 38

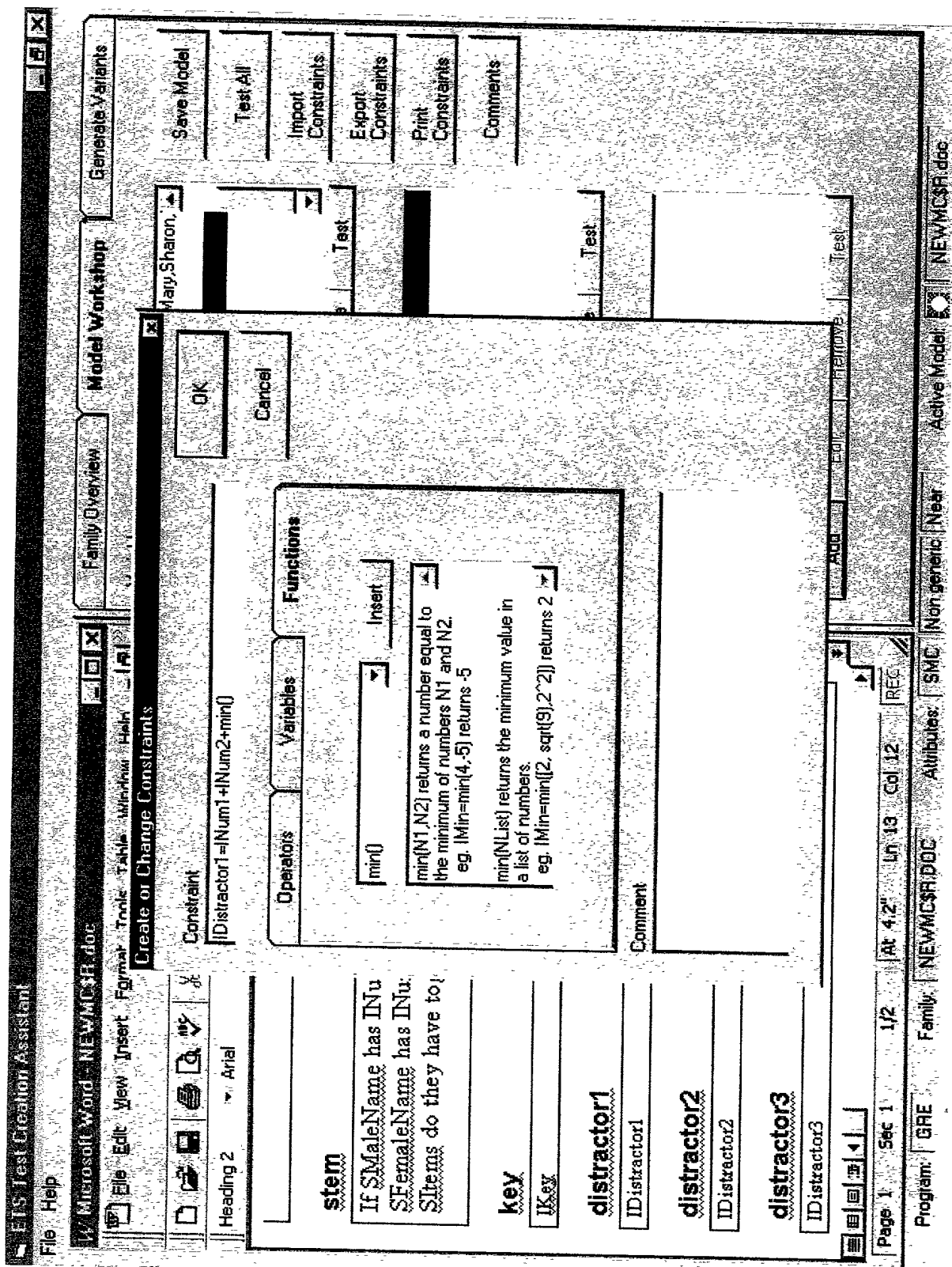


FIG. 39

Microsoft Word - NEWMCSR.doc

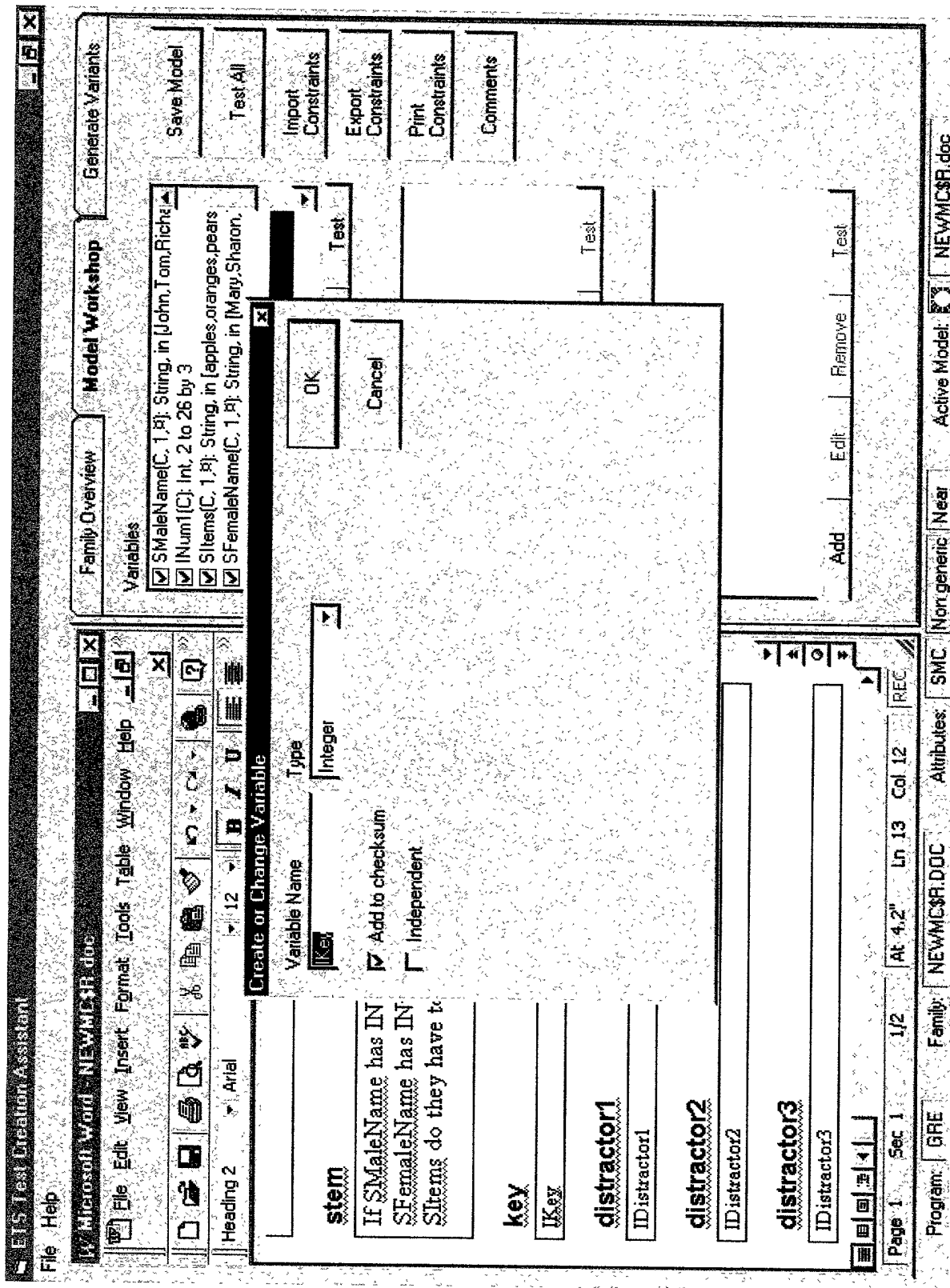


FIG. 28

Microsoft Word - NEWMC\$R.doc

EIS Test Creation Assistant

File Help

Microsoft Word - NEWMC\$R.doc

File Edit View Insert Format Tools Table Window Help

Heading 2 Arial 12

stem

If SMaleName has INum1 SItems and SFemaleName has INum2 SItems, how many SItems do they have together?

key

IKey

distractor1

IDistractor1

distractor2

IDistractor2

distractor3

IDistractor3

Page 1 Sec 1 1/2 At 4.2" Ln 13 Col 12 REC

Program: GRE Family: NEWMC\$R.DOC Attributes: SMC Non generic Near Active Model: NEWMC\$R.doc

Family Overview Model Workshop Generate Variants

Variables

☒ SFemaleName(C, 1, R): String, in [Mary, Sharon, ...]

☒ INum2(C): Int, 2 to 27 by 5

☒ IKey(C): Int

☒ IDistractor1(C): Int

☒ IDistractor2(C): Int

☒ IDistractor3(C): Int

☒ IDistractor4(C): Int

Add Edit Remove Test

Variation Constraints

Add Edit Remove Test

Save Model

Test All

Import Constraints

Export Constraints

Print Constraints

Comments

Distractor Constraints

Click here to add a variation constraint.

FIG. 29

Microsoft Word - NEWMC\$R.doc

ETS Test Creation Assistant

File Help

Microsoft Word - NEWMC\$R.doc

File Edit View Insert Format Tools Table Window Help

Create or Change Constraints

Constraint

Operators

Variables

Functions

SMaleName

SMaleName

INum1

SItems

SFemaleName

INum2

IKey

IDistractor1

IDistractor2

Insert

Comment

OK

Cancel

Family Overview

Model Workshop

Generate Variants

Variables

☒ SFemaleName(C, 18); String, in [Mary, Sharon, ...]

☒ INum2(C); Int, 2 to 27 by 5

☒ IKey(C); Int

☒ IDistractor1(C); Int

☒ IDistractor2(C); Int

☒ IDistractor3(C); Int

☒ IDistractor4(C); Int

Add

Edit

Remove

Test

Variation Constraints

Add

Edit

Remove

Test

Distractor Constraints

Add

Edit

Remove

Test

Save Model

Test All

Import Constraints

Export Constraints

Print Constraints

Comments

Page 1

Sec 1

1/2

At 4.2"

Ln 13

Col 12

REC

Program: GRE

Family: NEWMC\$R.DOC

Attributes: SMC

Non generic: Near

Active Model: NEWMC\$R.doc

FIG. 31

Microsoft Word - NEWMCSR.doc

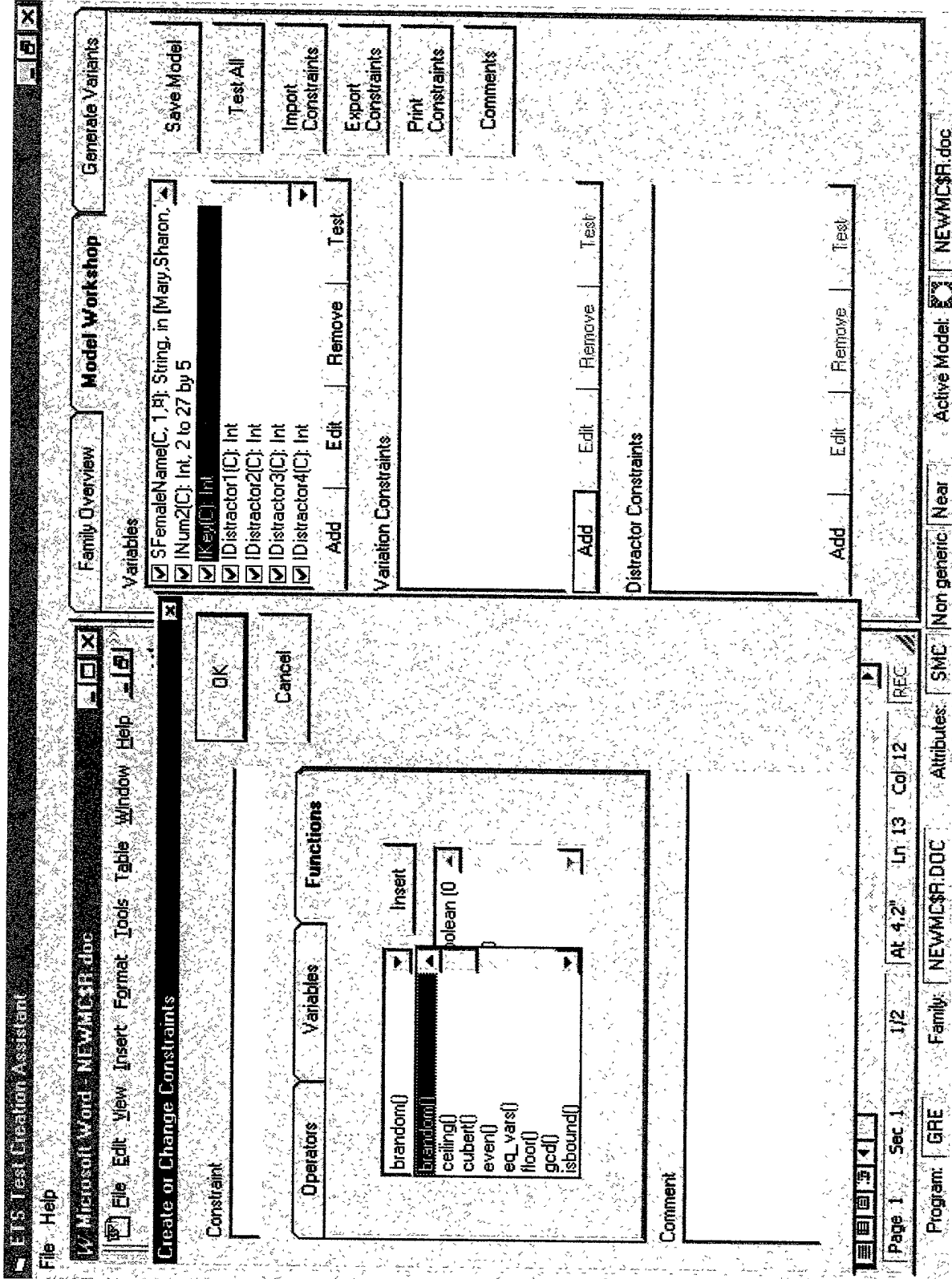


FIG. 32

Microsoft Word - NEWMC\$R.doc
File Edit View Insert Format Tools Table Window Help

ETS Test Creation Assistant

Microsoft Word - NEWMC\$R.doc

Create or Change Constraints

Constraint

Operators Variables Functions Insert

SMaleName
SMaleName
INum1
SItems
SFemaleName
INum2
IKey
IDistractor1
IDistractor2

Comment

OK Cancel

Family Overview Model Workshop Generate Variants

Variables

☒ SFemaleName(C, 1, 8): String, in Mary.Sharon, A
☒ INum2(C): Int, 2 to 27 by 5
☒ IKey(C): Int
☒ IDistractor1(C): Int
☒ IDistractor2(C): Int
☒ IDistractor3(C): Int
☒ IDistractor4(C): Int

Add Edit Remove Test

Variation Constraints

Add Edit Remove Test

Distractor Constraints

Add Edit Remove Test

Save Model
Test All
Import Constraints
Export Constraints
Print Constraints
Comments

Page 1 Sec 1 1/2 At 4.2 Ln 13 Col 12 REC

Program: GRE Family: NEWMC\$R.DOC Attributes: SMC Non generic: Near Active Model: NEWMC\$R.doc

FIG. 33

Microsoft Word - NEWMCSR.doc

File Help

Microsoft Word - NEWMCSR.doc

File Edit View Insert Format Tools Table Window Help

Create or Change Constraints

Constraint

|Key

Operators

Variables

Functions

Comment

OK

Cancel

EIS Test Creation Assistant

Family Overview

Model Workshop

Generate Variants

Variables

☒ SFemaleName(C, 1, 8): String, in [Mary, Sharon],

☒ INum2(C): Int, 2 to 27 by 5

☒ |Key(C): Int

☒ IDistractor1(C): Int

☒ IDistractor2(C): Int

☒ IDistractor3(C): Int

☒ IDistractor4(C): Int

Add

Edit

Remove

Test

Variation Constraints

Add

Edit

Remove

Test

Distractor Constraints

Add

Edit

Remove

Test

Save Model

Test All

Import Constraints

Export Constraints

Print Constraints

Comments

Page 1

Sec 1

1/2

At 4.2"

Ln 13

Col 12

REC

Program: GRE

Family: NEWMCSR.DOC

Attributes: SMC

Non generic

Near

Active Model: NEWMCSR.doc

FIG. 34

FIG. 35

Microsoft Word - NEWMCSR.doc

EIS Test Creation Assistant

File Help

Microsoft Word - NEWMCSR.doc

File Edit View Insert Format Tools Table Window Help

Create or Change Constraints

Constraint

Operators

Variables

Functions

Insert

Comment

OK

Cancel

Family Overview

Model Workshop

Generate Variants

Variables

☒ SFemaleName(C, 1, 8): String, in [Mary, Sharon],

☒ Num2(C): Int, 2 to 27 by 5

☒ Key(C): Int

☒ Distractor1(C): Int

☒ Distractor2(C): Int

☒ Distractor3(C): Int

☒ Distractor4(C): Int

Add

Edit

Remove

Test

Variation Constraints

Add

Edit

Remove

Test

Distractor Constraints

Add

Edit

Remove

Test

Save Model

Test All

Import Constraints

Export Constraints

Print Constraints

Comments

Page 1

Sec 1

1/2

At 4.2"

Ln 13

Col 12

REC

Program: GRE

Family: NEWMCSR.DOC

Attributes: SMC

Non generic

Near

Active Model: NEWMCSR.doc

FIG. 36

FIG. 37

FIG. 38

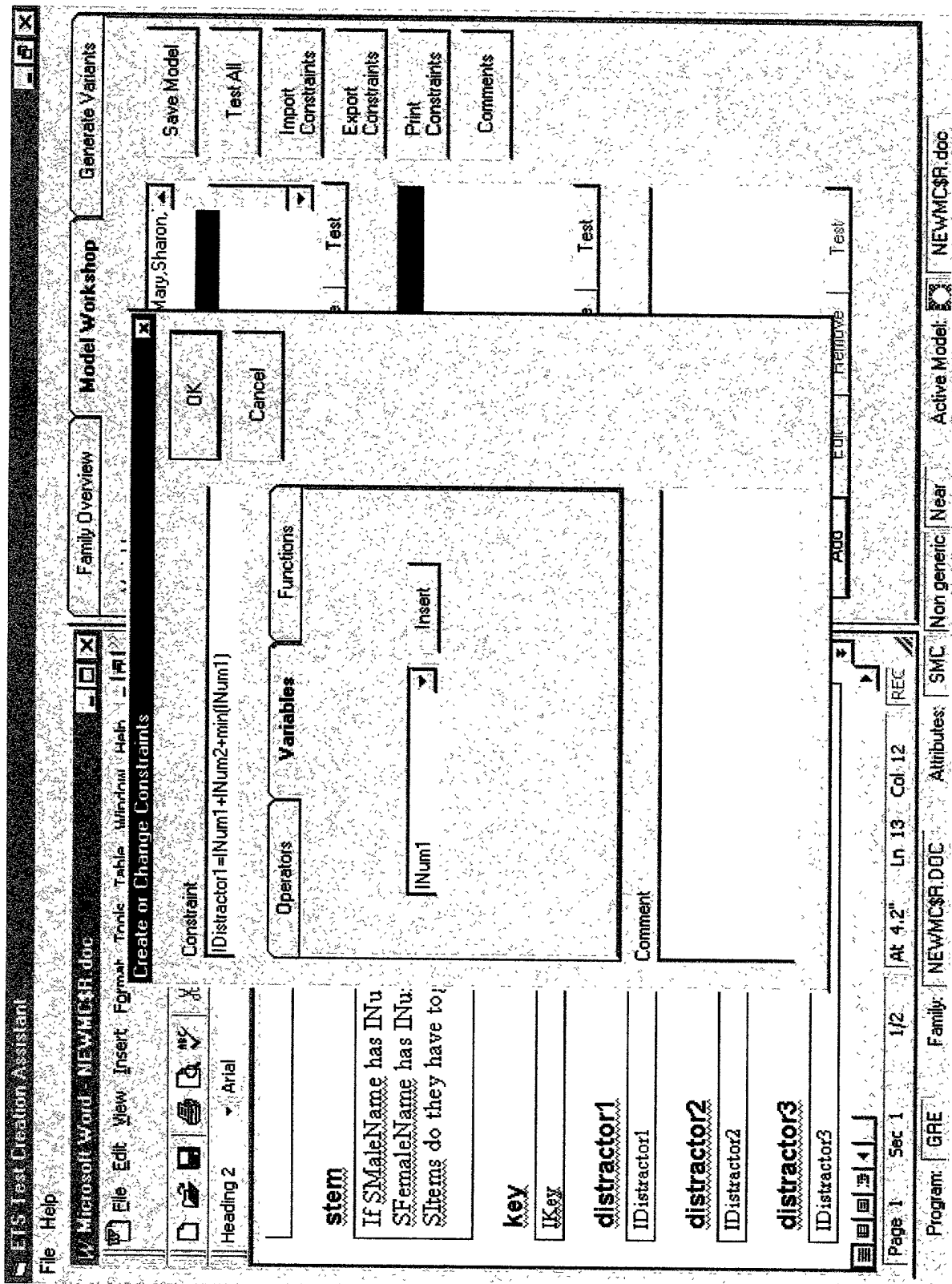


FIG. 40

Microsoft Word - NEWMSR.doc

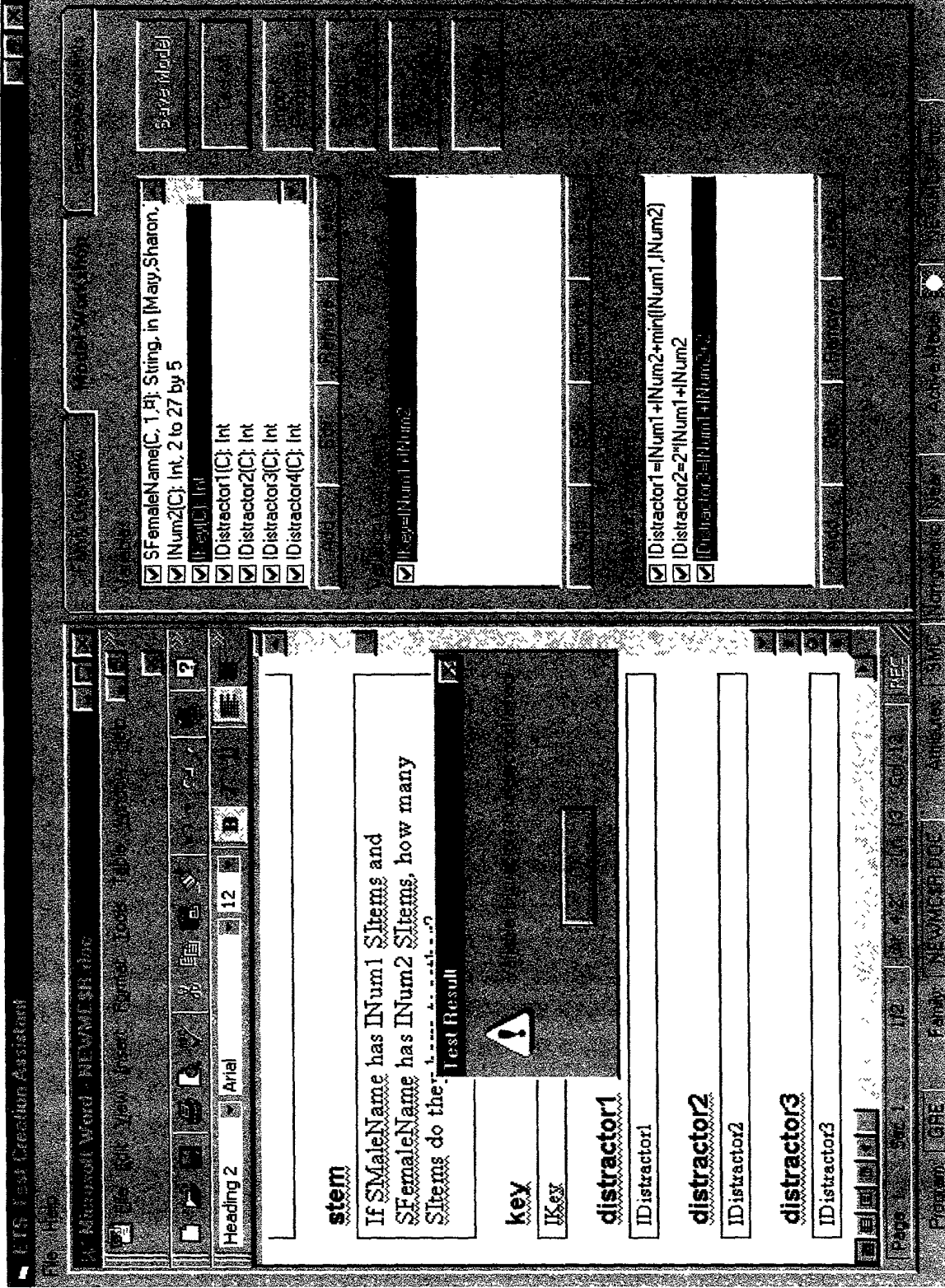


FIG. 43

Small text or logo at the top of the page, possibly a page number or header.

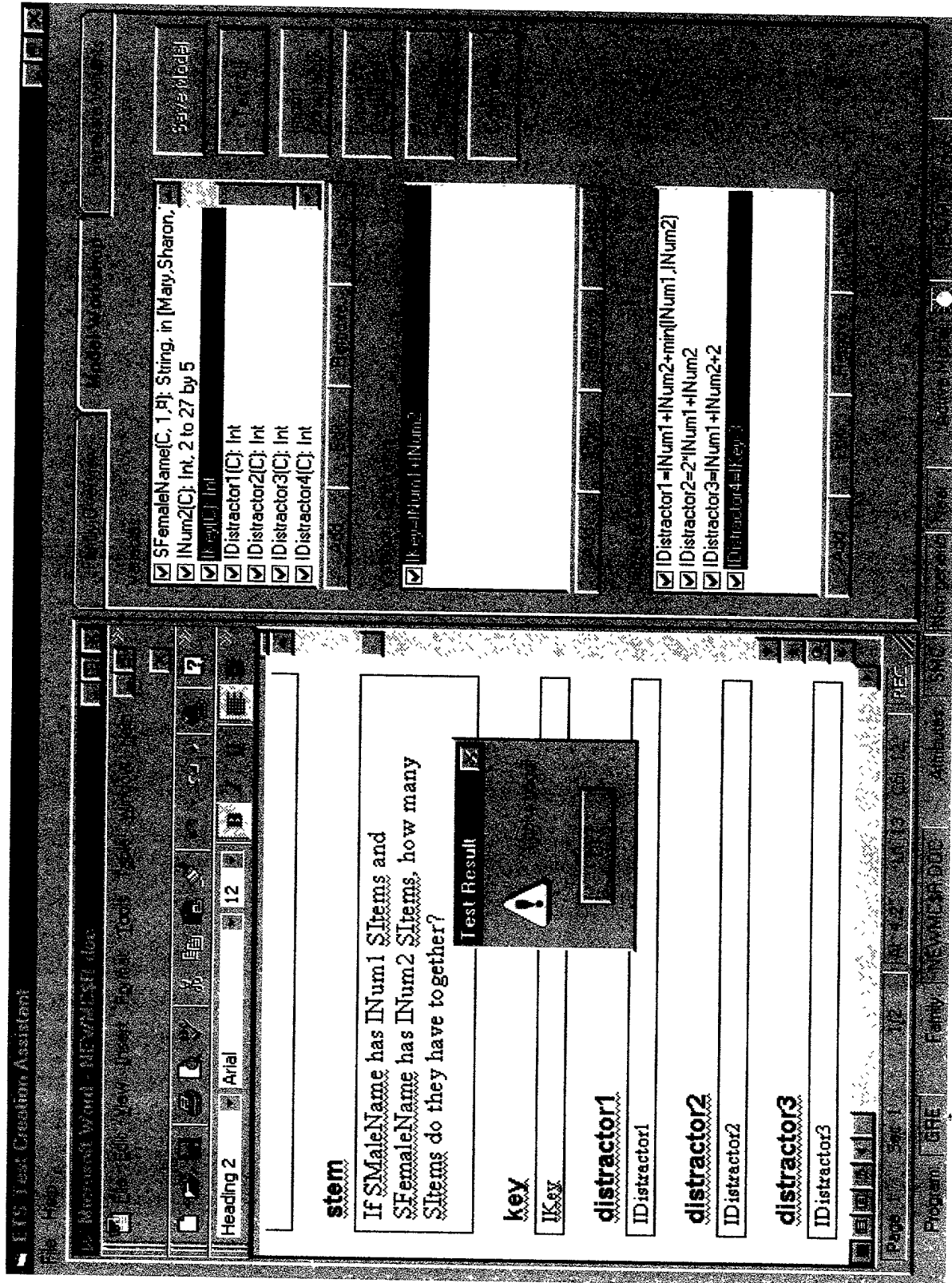


FIG. 44

[illegible]

FIG. 45



FD-48

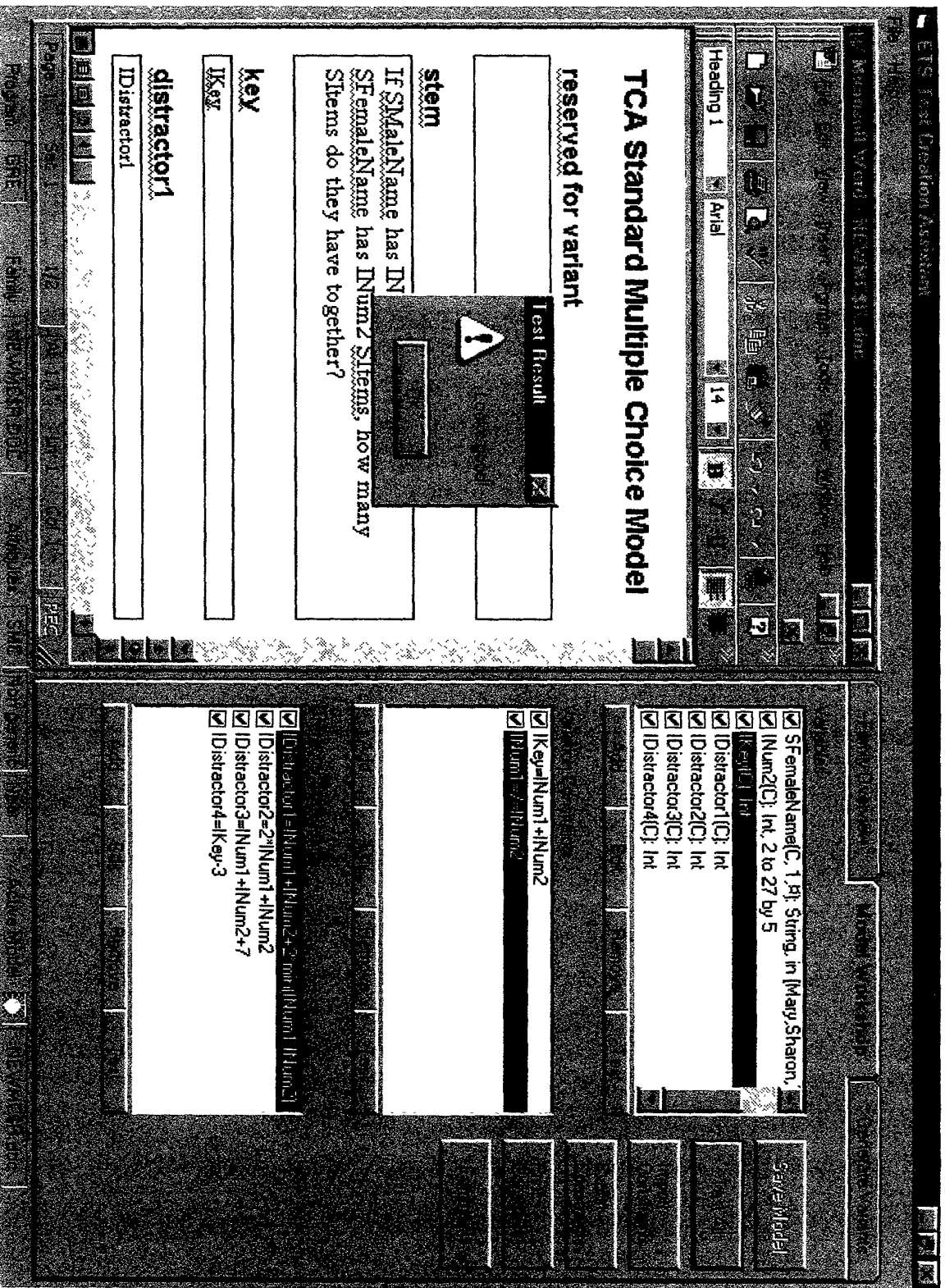


FIG. 49

FIG. 50

FIG. 50

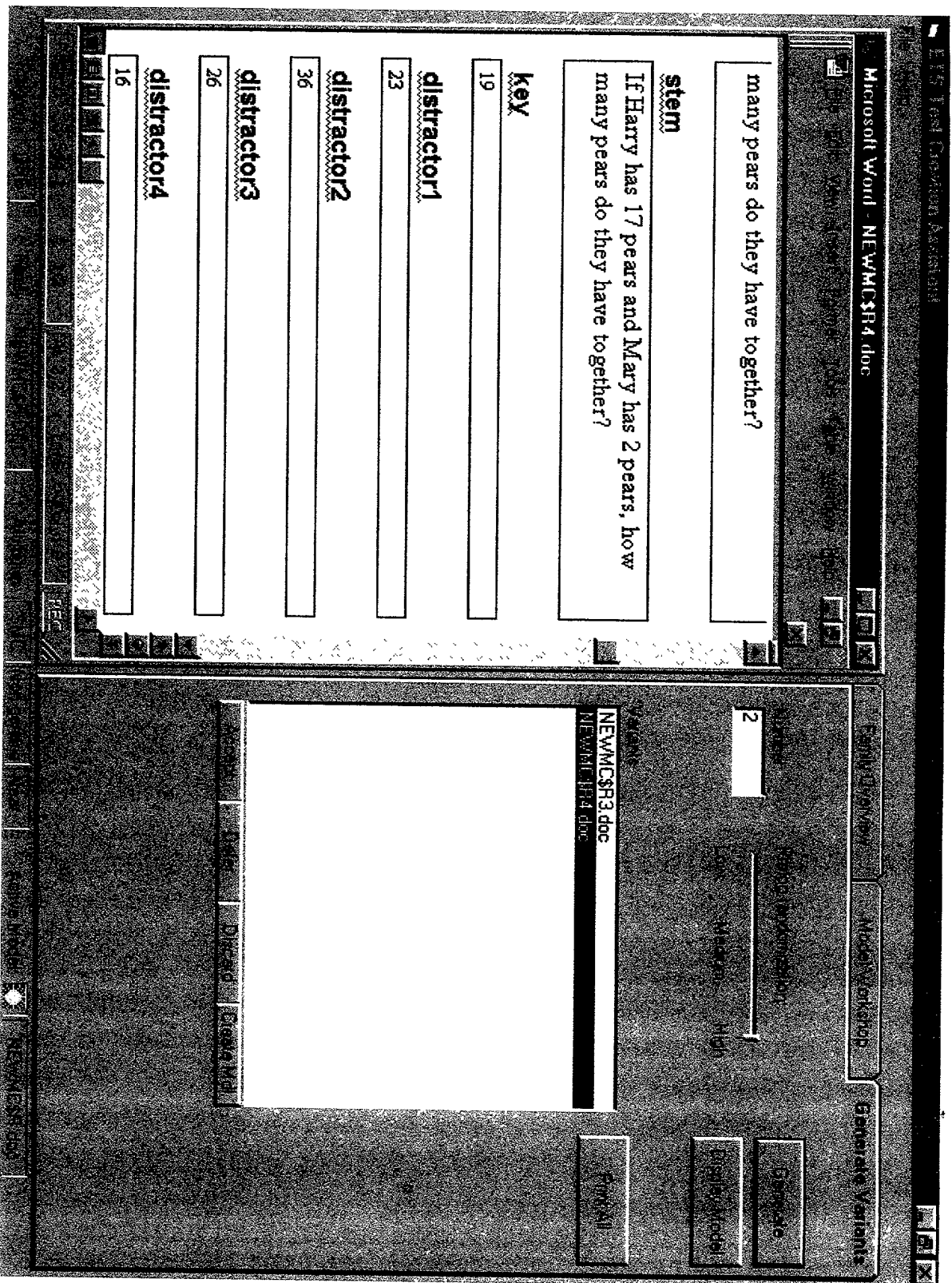


FIG. 51

EIS Test Creation Assistant

File Edit View Insert Format Tools Help Window Help

many pears do they have together?

stem

If Harry has 17 pears and Mary has 2 pears, how many pears do they have together?

key

19

distractor1

23

distractor2

36

distractor3

26

distractor4

16

Click here to accept the currently selected variants.

Number: 2

Number of responses: 1

Low Medium High

NEWMC\$R3.doc
NEWMC\$R4.doc

Generate Variants

Generate

Display Model

Print All

Answer

Blank

Record

Display

Answer Model

NEWMC\$R3.doc

FIG. 52



FIG. 53



1

Year	Number of people (millions)
1980	20
1985	22
1990	24
1995	26
2000	28
2005	30
2010	32
2015	34
2020	38

1

25



Abstract

ad variant

Corporate Finance

以爲第一

100

100

五

全

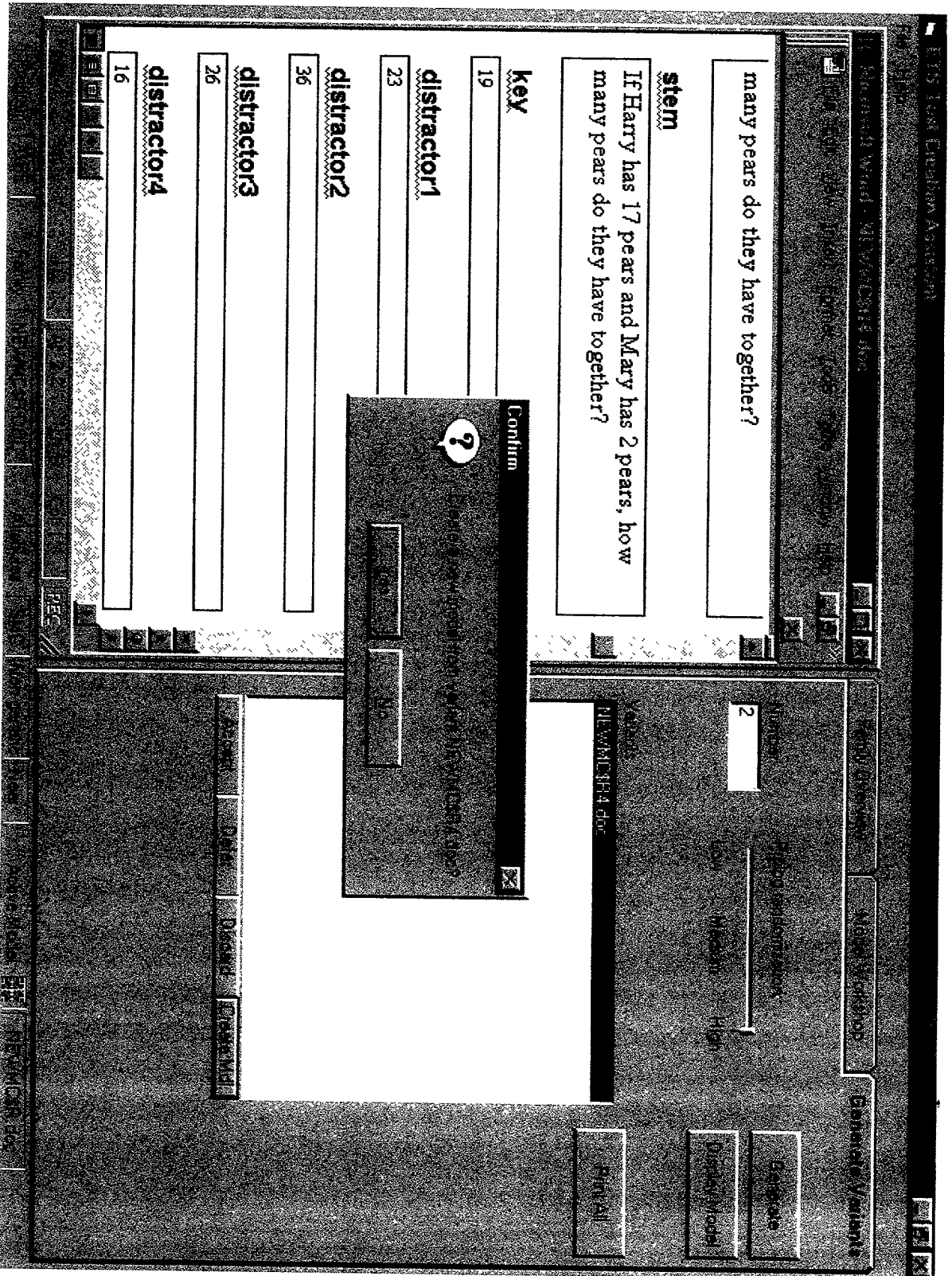


FIG. 55

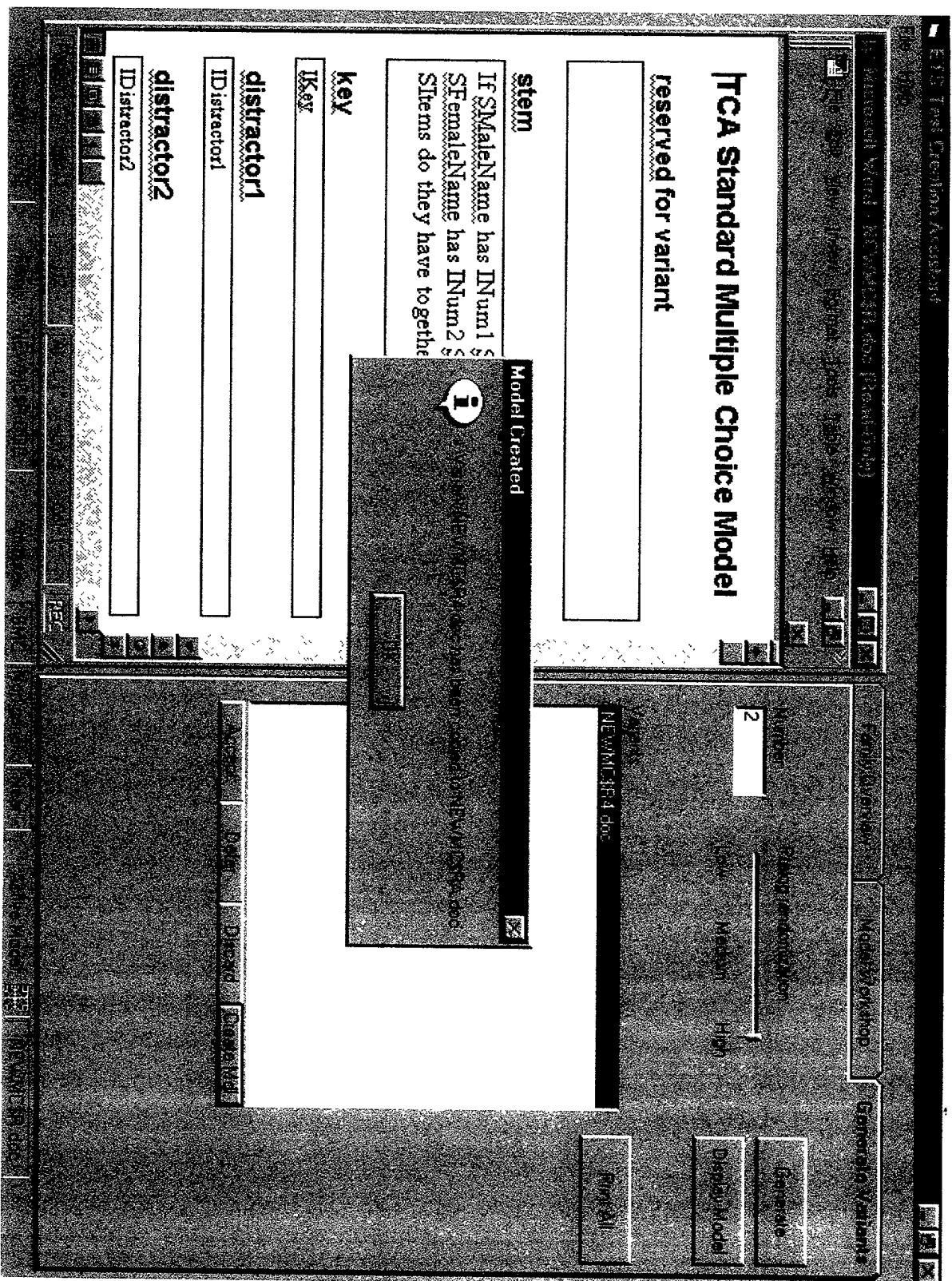


FIG. 56

ETS Test Creation Assistant

Microsoft Word 97 (File Edit Format Tools Window Help)

Heading 1 Arial 14

TCA Standard Multiple Choice Model

reserved for variant

stem

If SMaleName has INum1 \$Items and SEemaleName has INum2 \$Items, how many \$Items do they have together?

key

IKey

distractor1

IDistractor1

Form Information

NEWMC\$RA.doc

NEWMC\$PB.doc

Set Attributes

Done

Standard Remove

Preview

FIG. 57

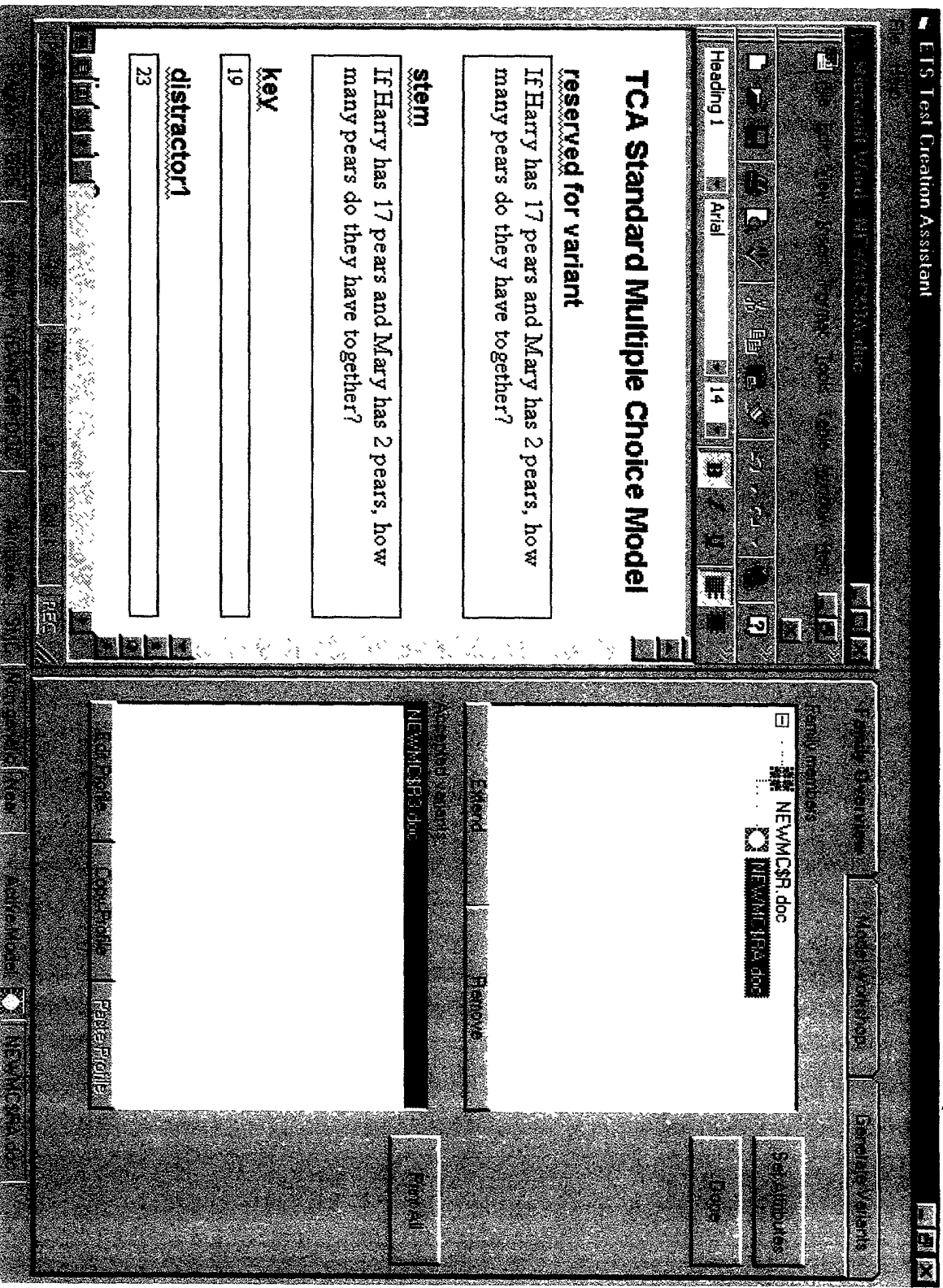


FIG. 58

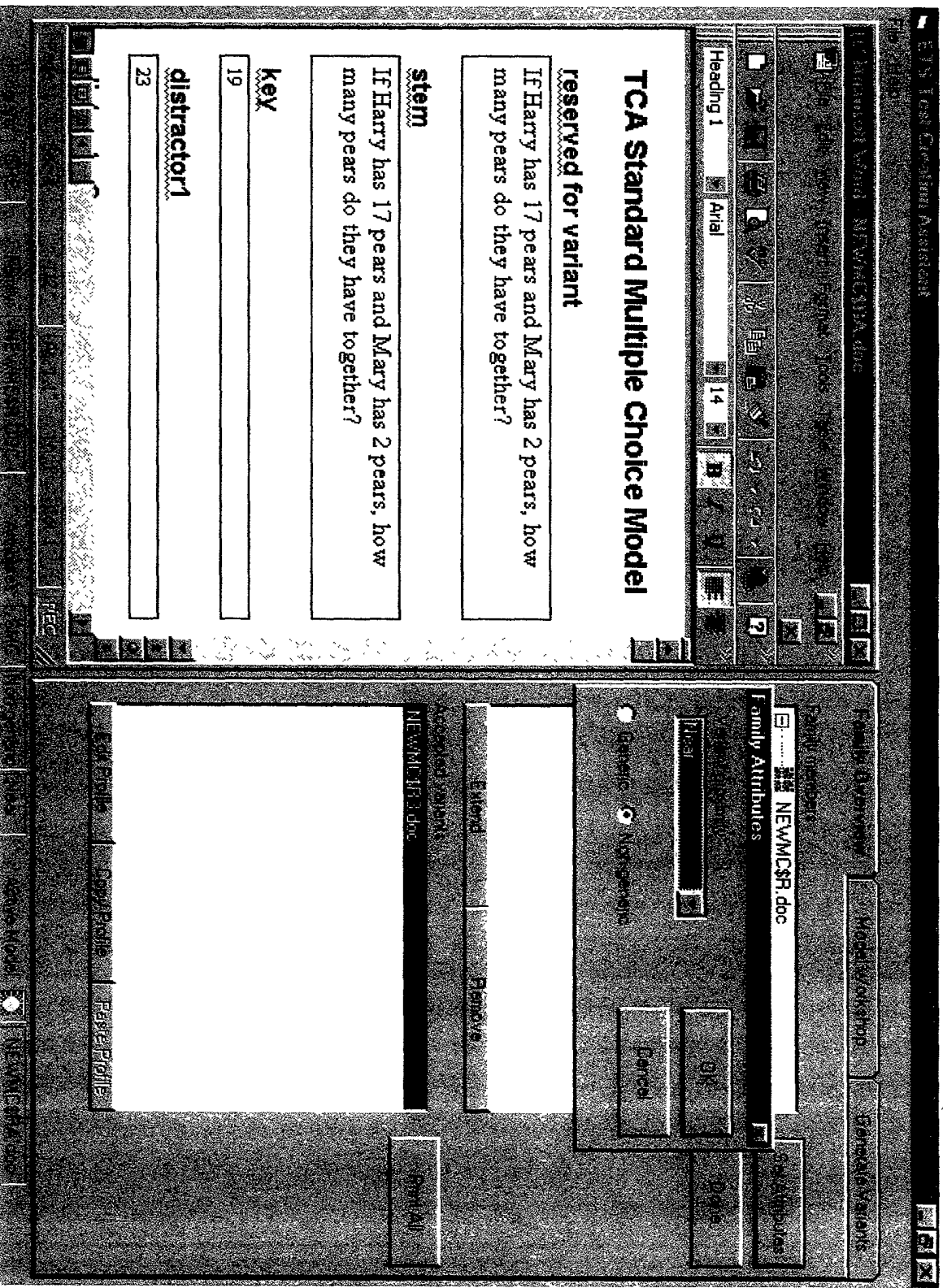


FIG. 59

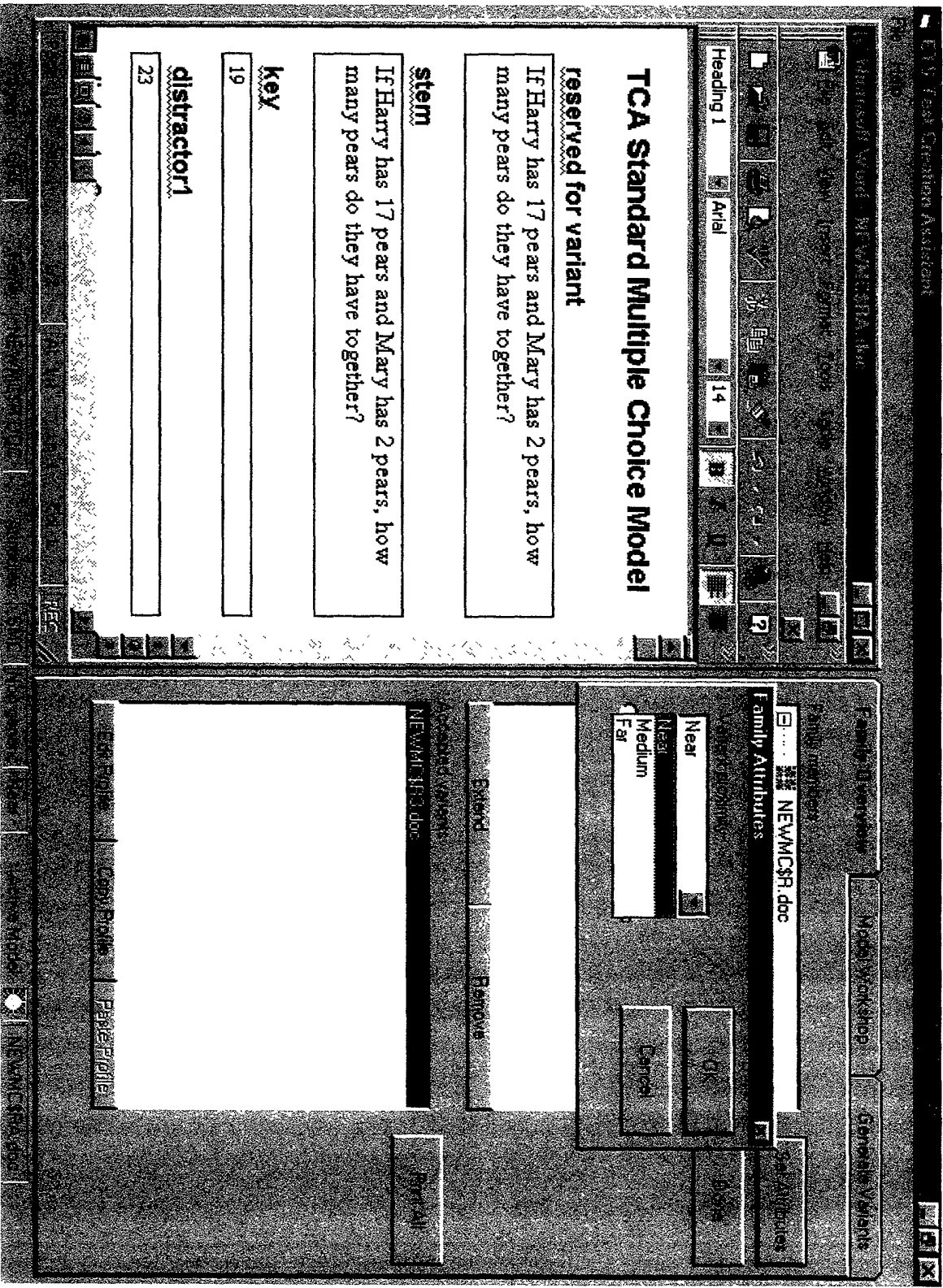


FIG. 60

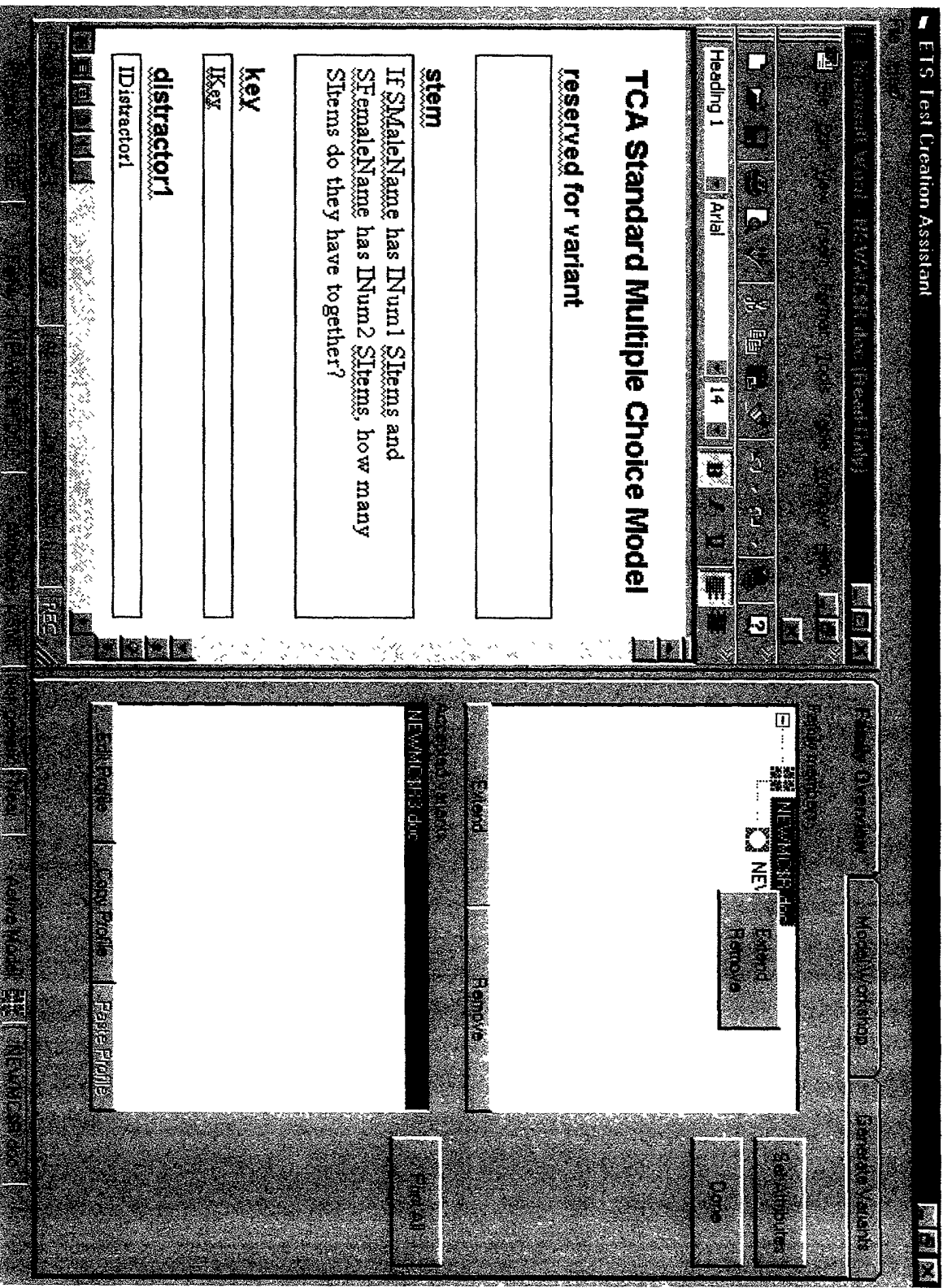


FIG. 61

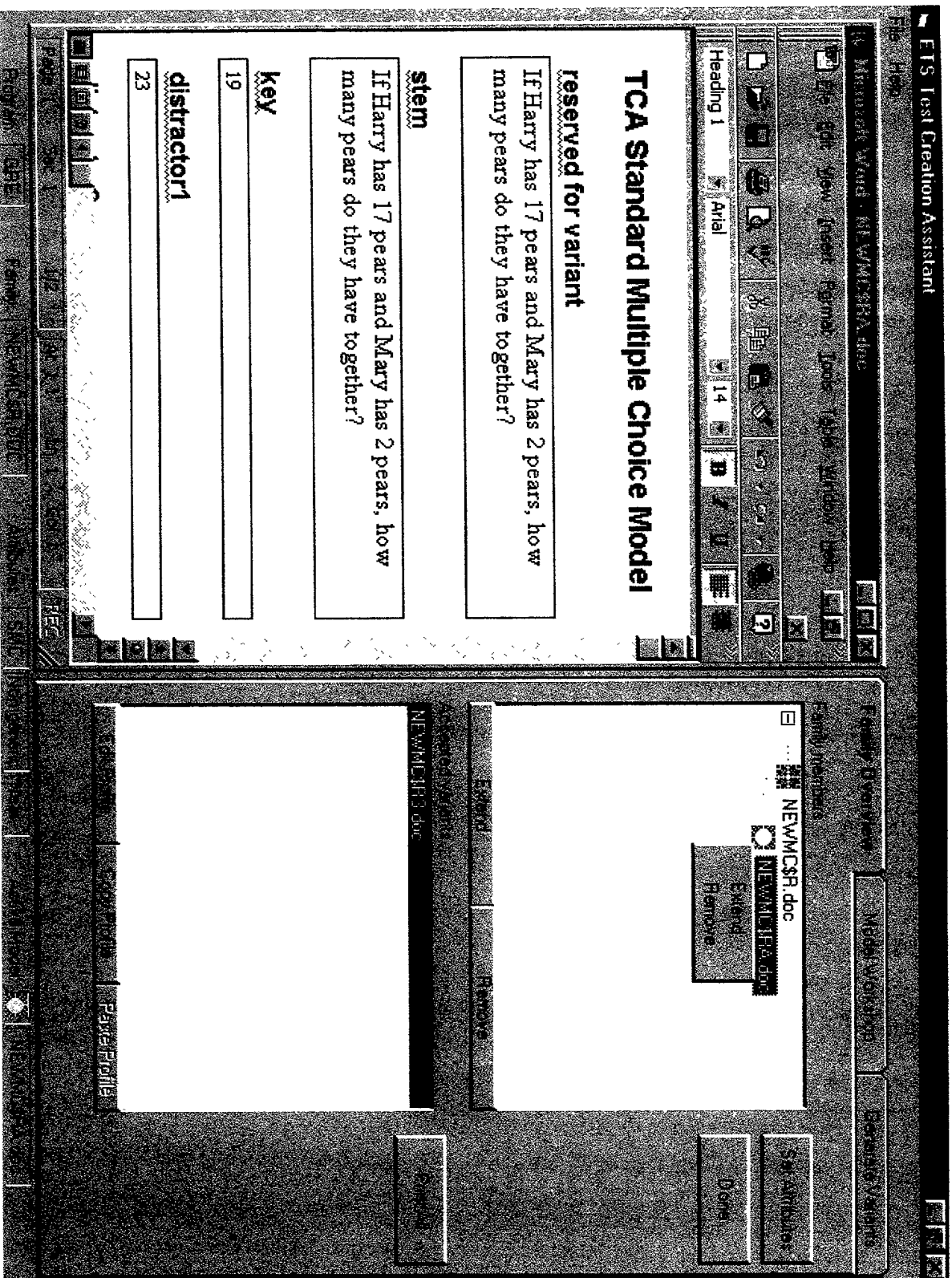


FIG. 62

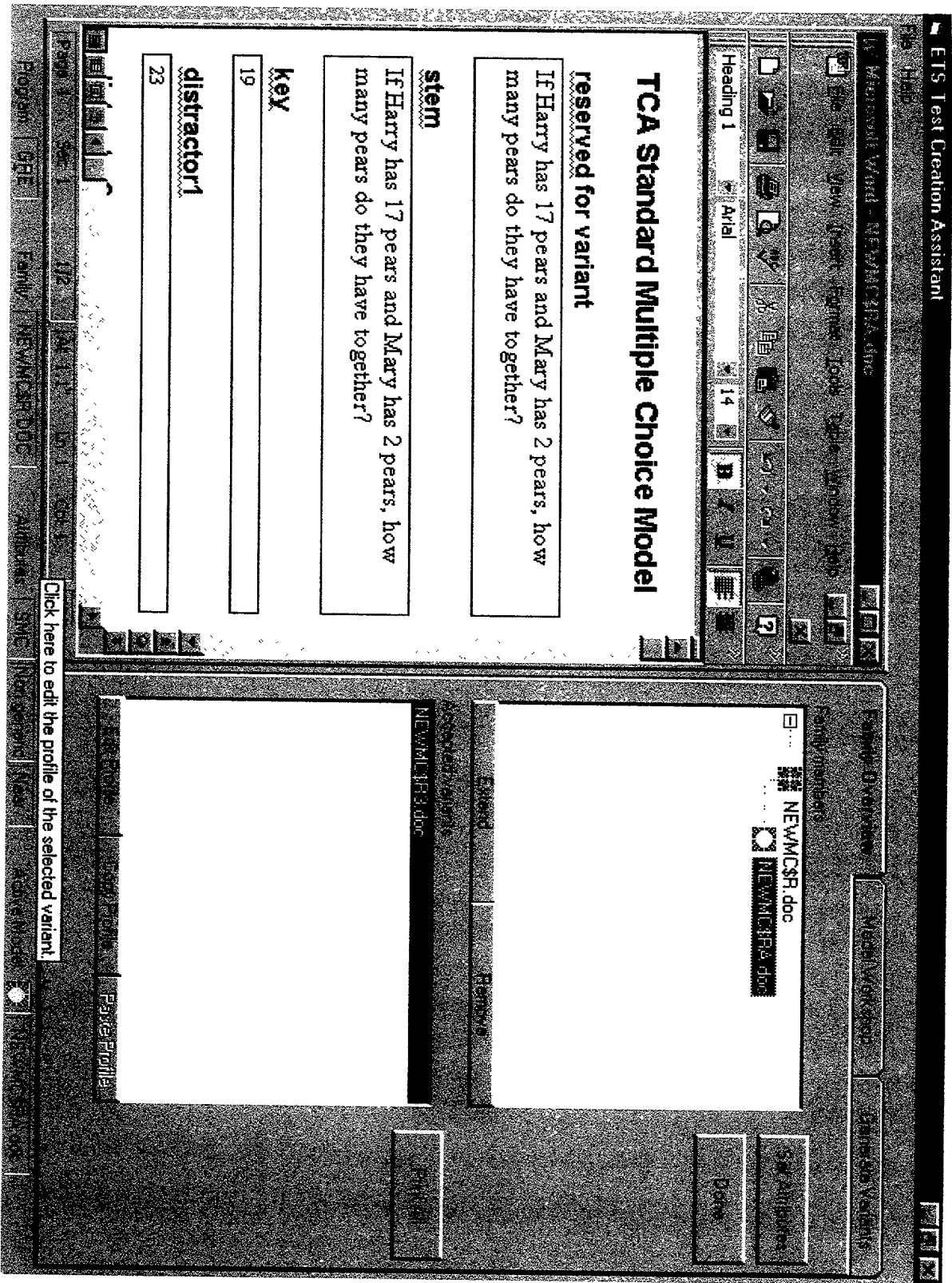


FIG. 63

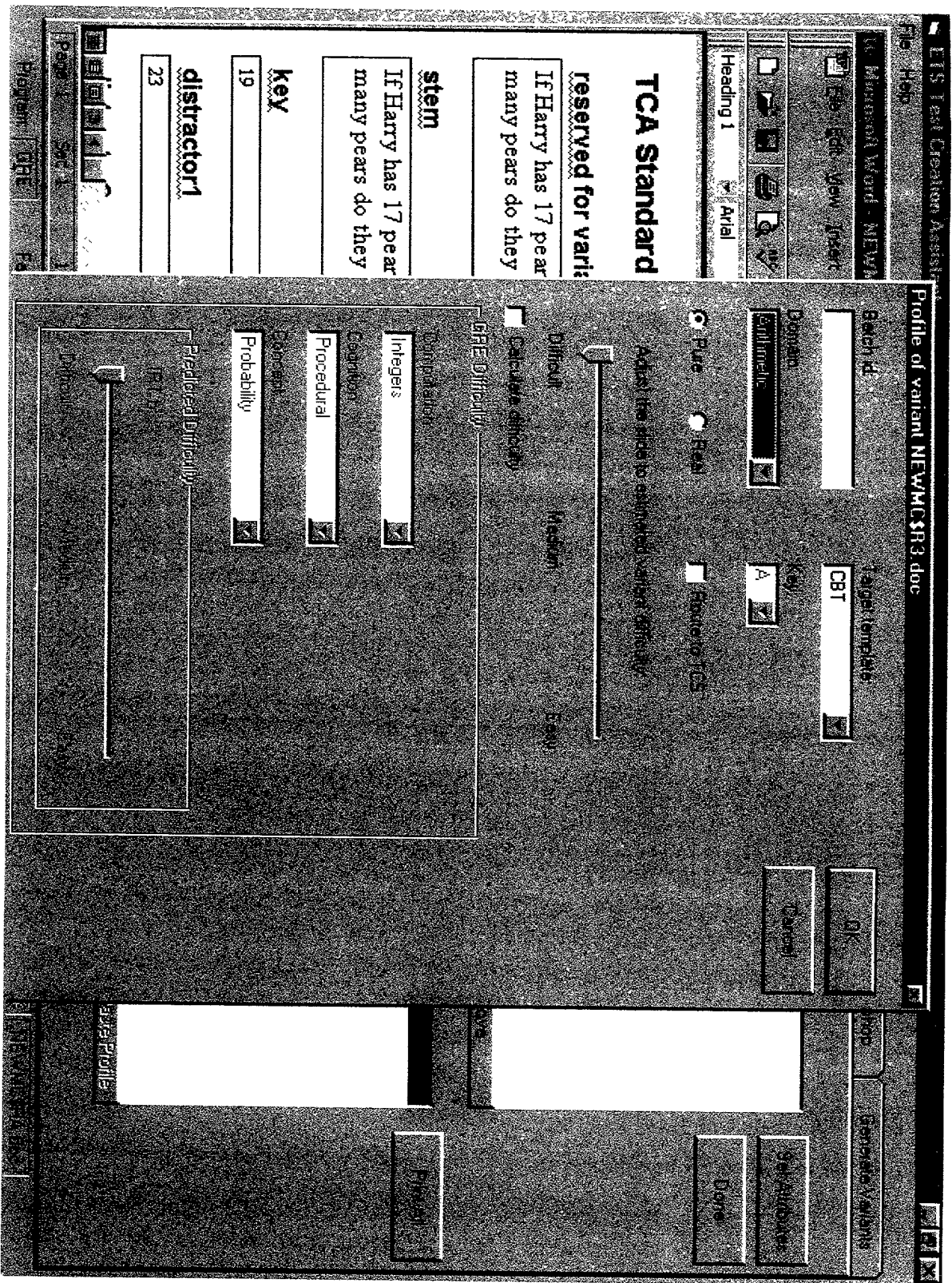


FIG. 64

ETS Test Creation Assistant

File Help

Microsoft Word - NEWMC\$R3.doc

File Edit View Insert

Heading 1 Arial

TCA Standard

reserved for vari

If Harry has 17 pear
many pears do they

stem

If Harry has 17 pear
many pears do they

key

19

distractor1

23

Page 1 of 1

Program GRE

Profile of variant NEWMC\$R3.doc

Baseid

CBT

Key

A

Domain

Arithmetic

Algebra

Data Analysis

Geometry

Adjust the slide to estimated variant difficulty

Difficult Medium Easy

☐ Calculate difficulty

GRE Difficulty

Completion

Integers

Cognition

Procedural

Concepts

Probability

Reduced Difficulty

Difficulty

OK

Cancel

Save Variants

Save Variants

Done

Profile

ETS Test Creation Assistant

File Edit View Insert

Heading 1 Arial

TCA Standard

reserved for vari:

If Harry has 17 pear
many pears do they

stem

If Harry has 17 pear
many pears do they

key

19

distractor1

23

Page 1 of 1

Profile of variant NEWMC\$R3.doc

Batchid: [] Target template: CBT []

Domain: Arithmetic [] PPI: A []

☐ Pure ☐ Mixed ☐ Made to fit

Adjust the slider to estimate variant difficulty

Difficulty: [] Medium [] Easy

☐ Calculator difficulty

EFHE Difficulty

Completion: Integers []

Cognition: Procedural []

Concept: Probability []

Predicted Difficulty: []

OK Cancel

Details Version

Save Cancel

FIG. 66

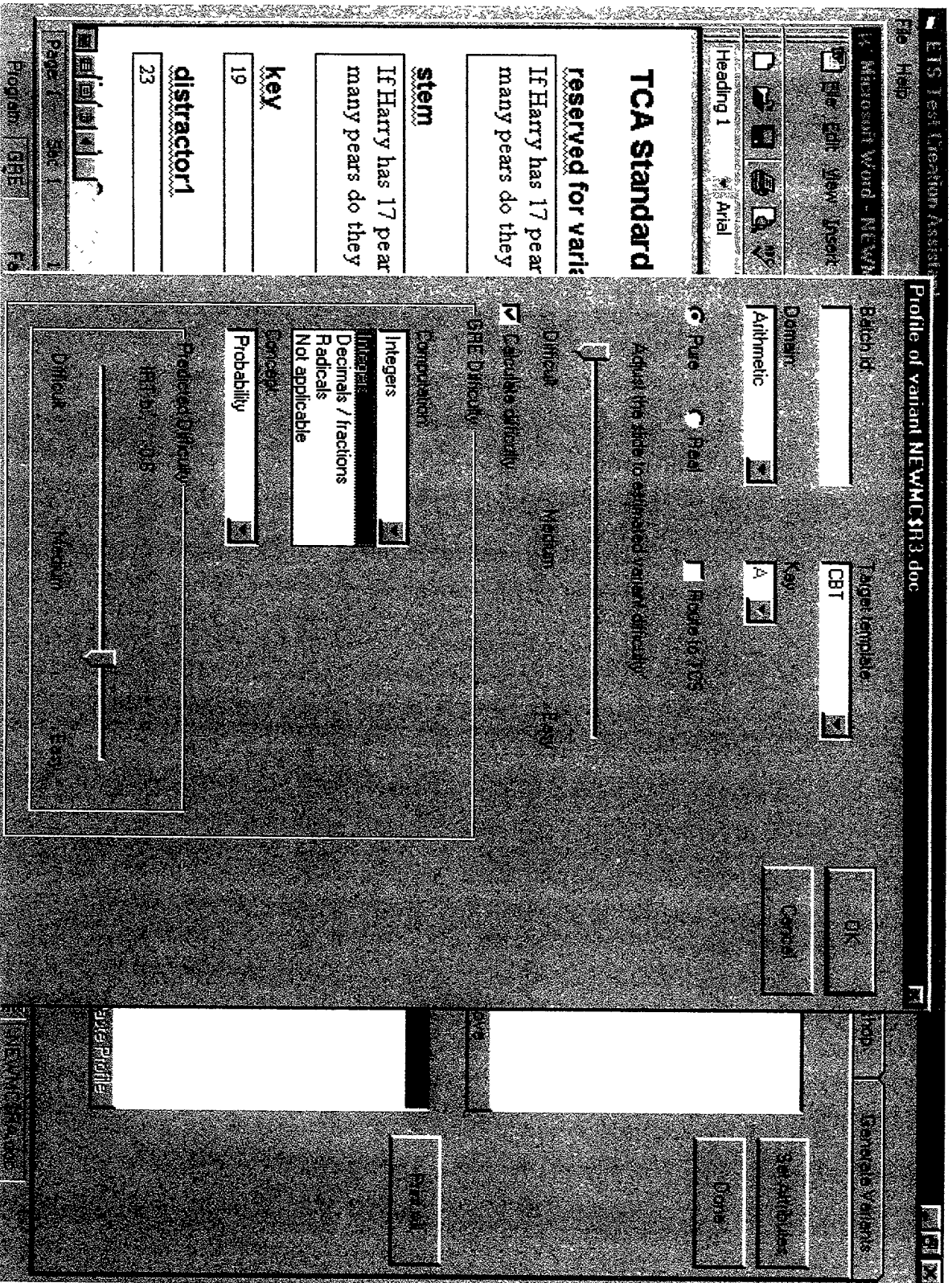


FIG. 68



ETS Test Creation Assistant

File Help

Microsoft Word - NEWM3H3.doc

File Edit View Insert Format Tools Window Help

Heading 1 Arial

TCA Standard

reserved for vari:

If Harry has 17 pear
many pears do they

stem

If Harry has 17 pear
many pears do they

key

19

distractor1

23

Page 1 of 1

Program GRE

Profile of variant NEWM3H3.doc

Batch ID: Target ID: CBT

Domain: Arithmetic Key: A

☐ Pure ☐ Real ☐ Random SS

Adjust the slope to estimate target difficulty

Difficulty: Medium Easy

☒ Calculate difficulty

IRE Difficulty:

Computation: Integers

Equation: Procedural

Concept: Probability

Probability:

Percent of a percent

Percent change

Linear inequality

Not applicable

Concept:

Difficulty:

Generate Variants

9th Attributes: Date:

Preview

FIG. 70

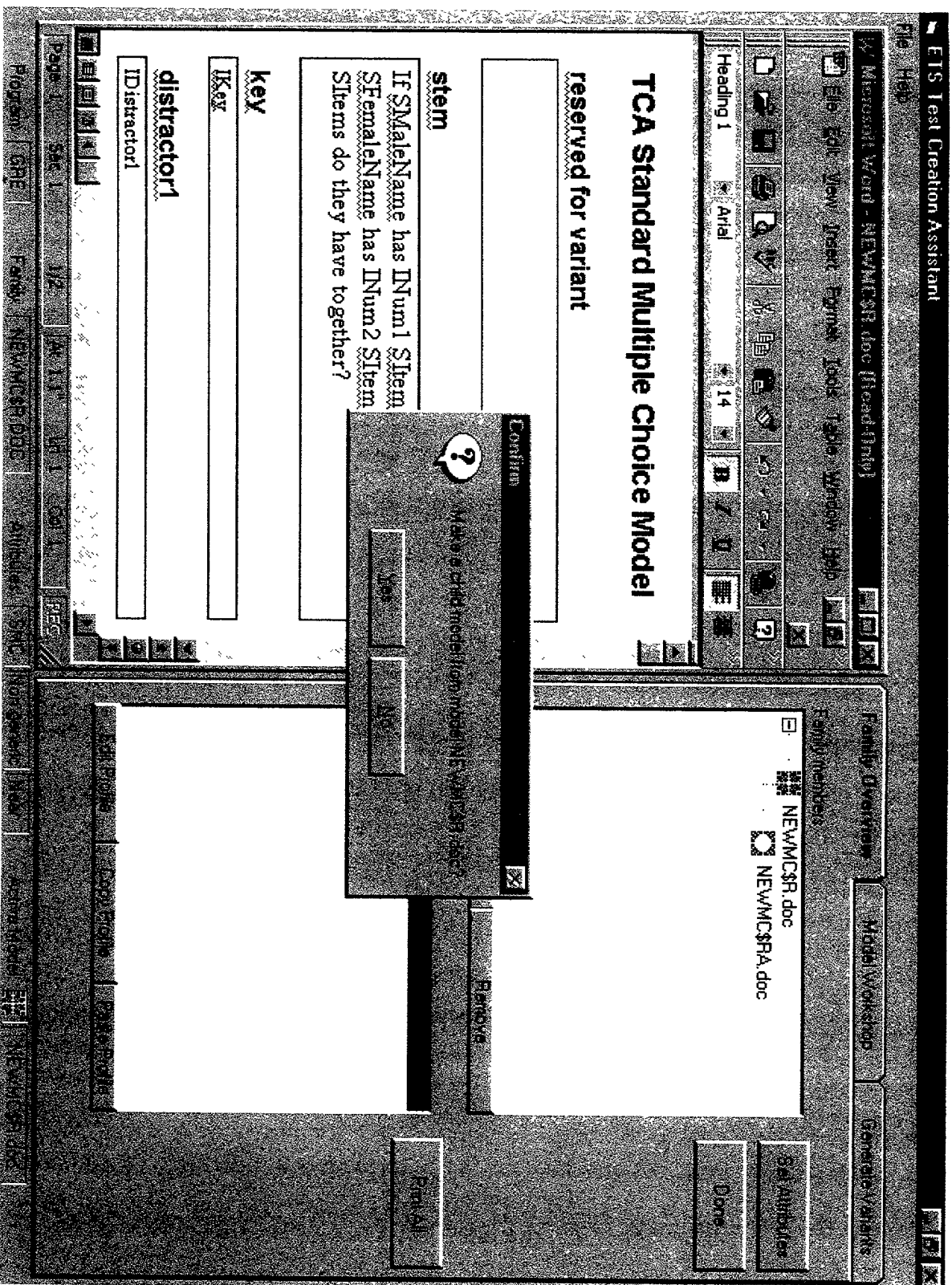


FIG. 72



FIG. 73

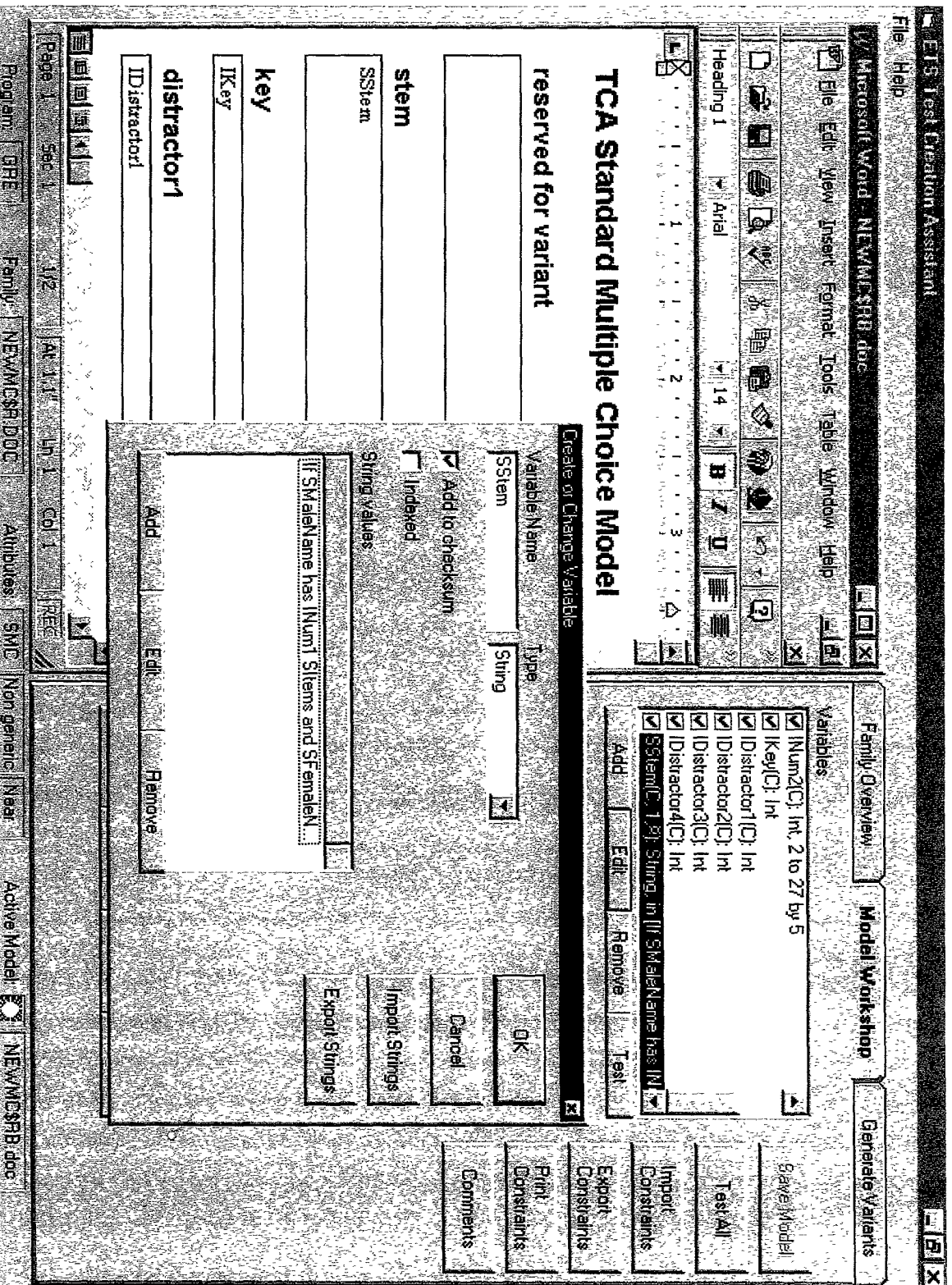


FIG. 73A

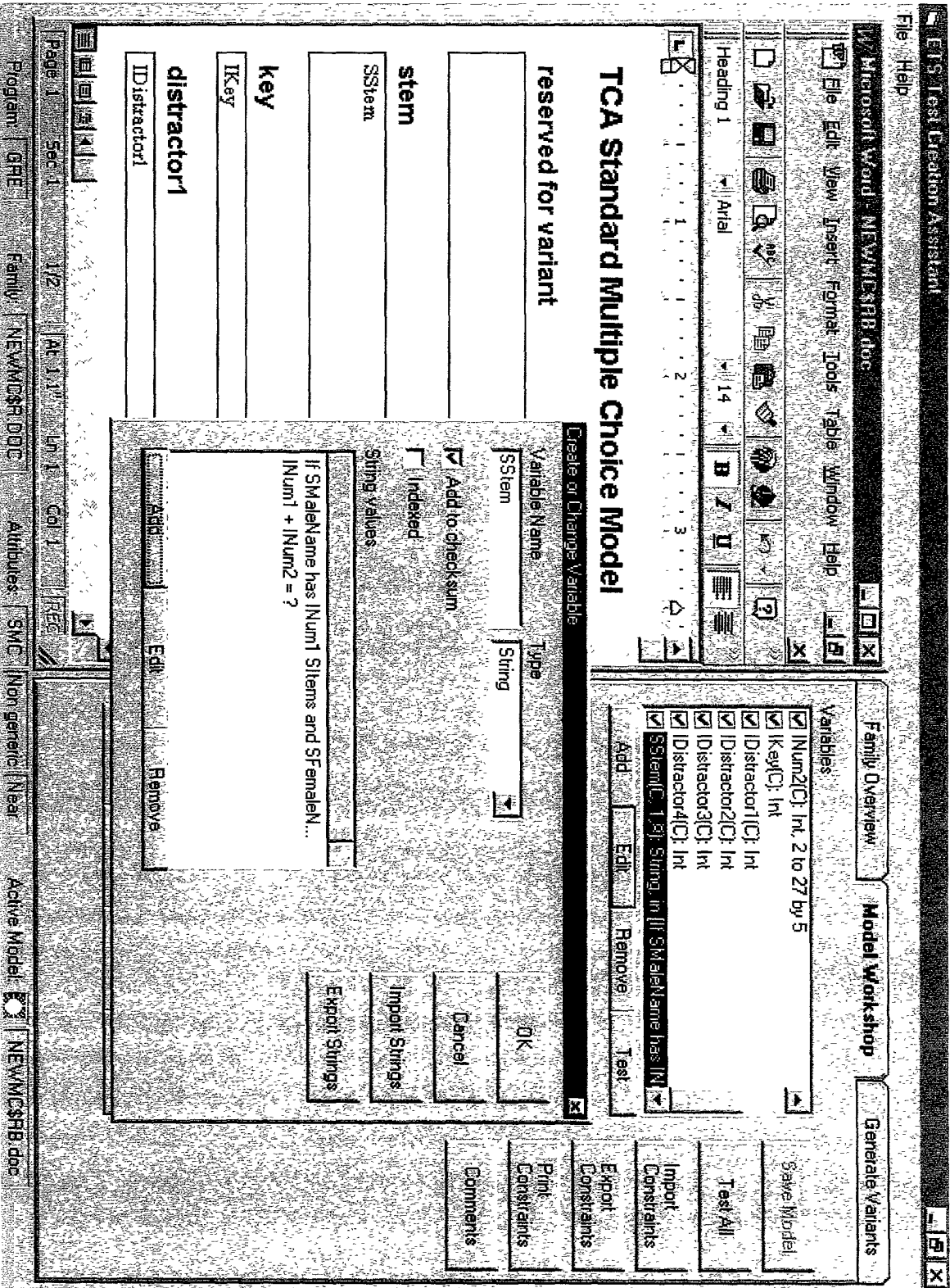


FIG. 73B

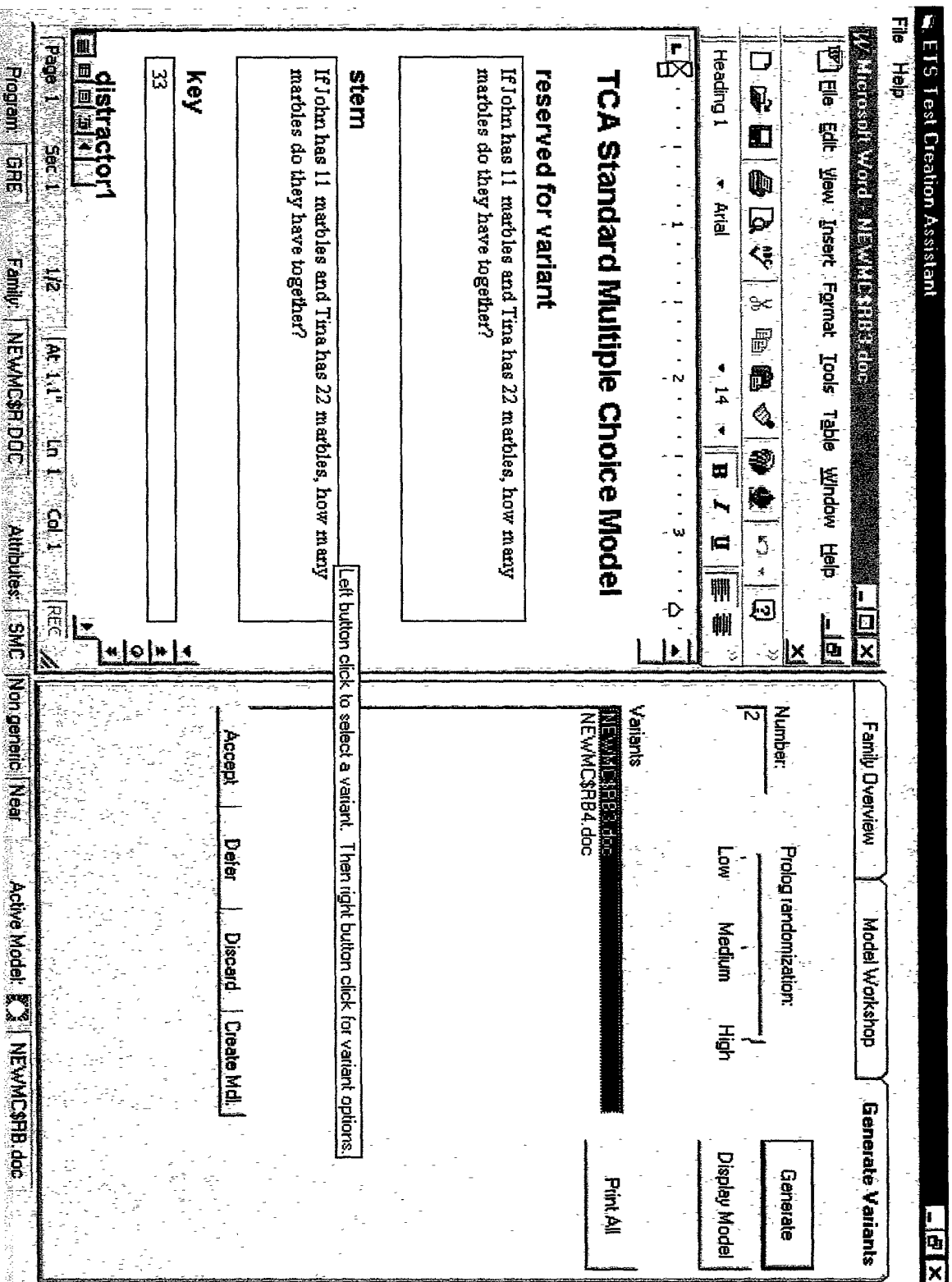


FIG. 73D

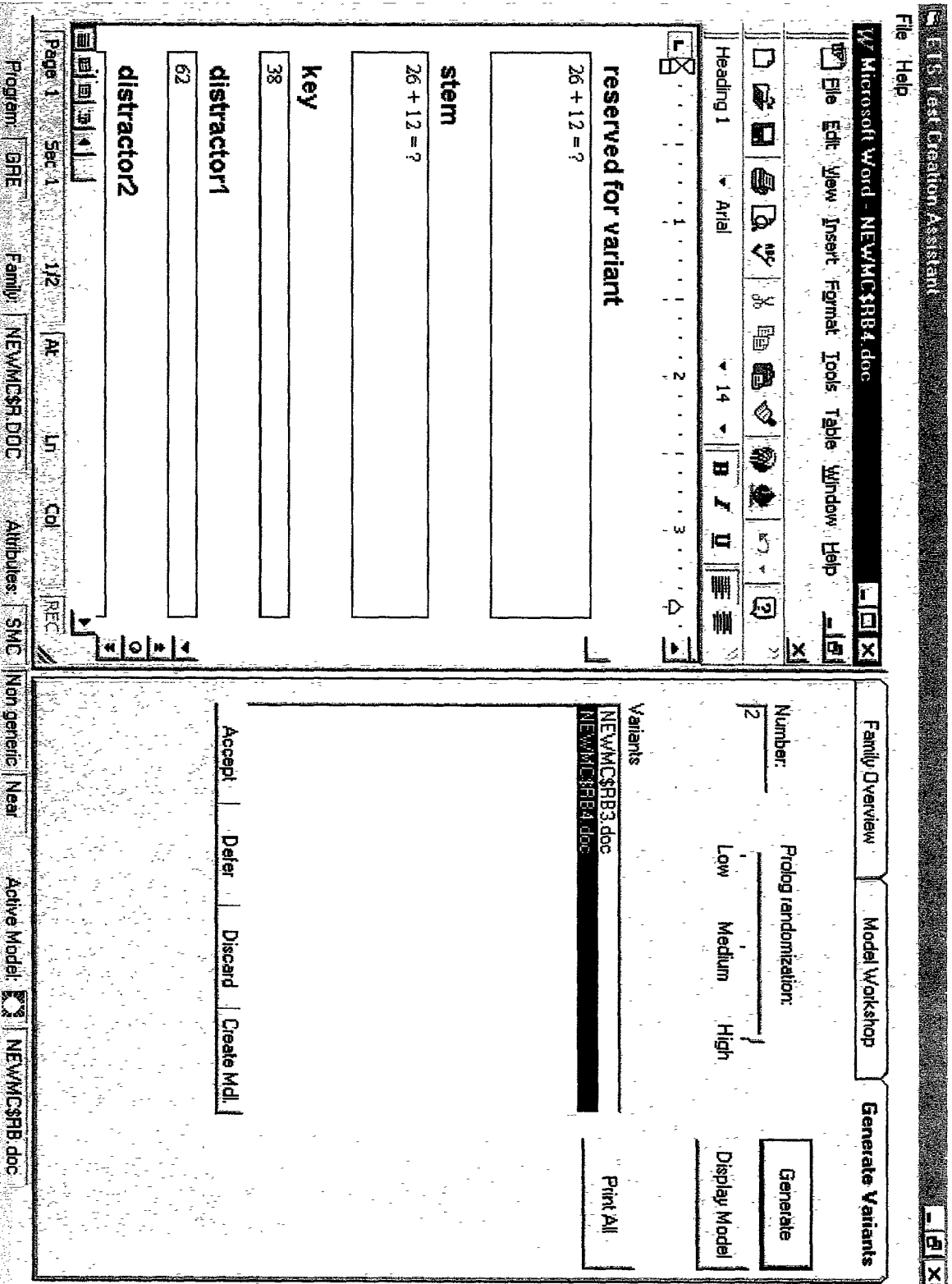


FIG. 73E

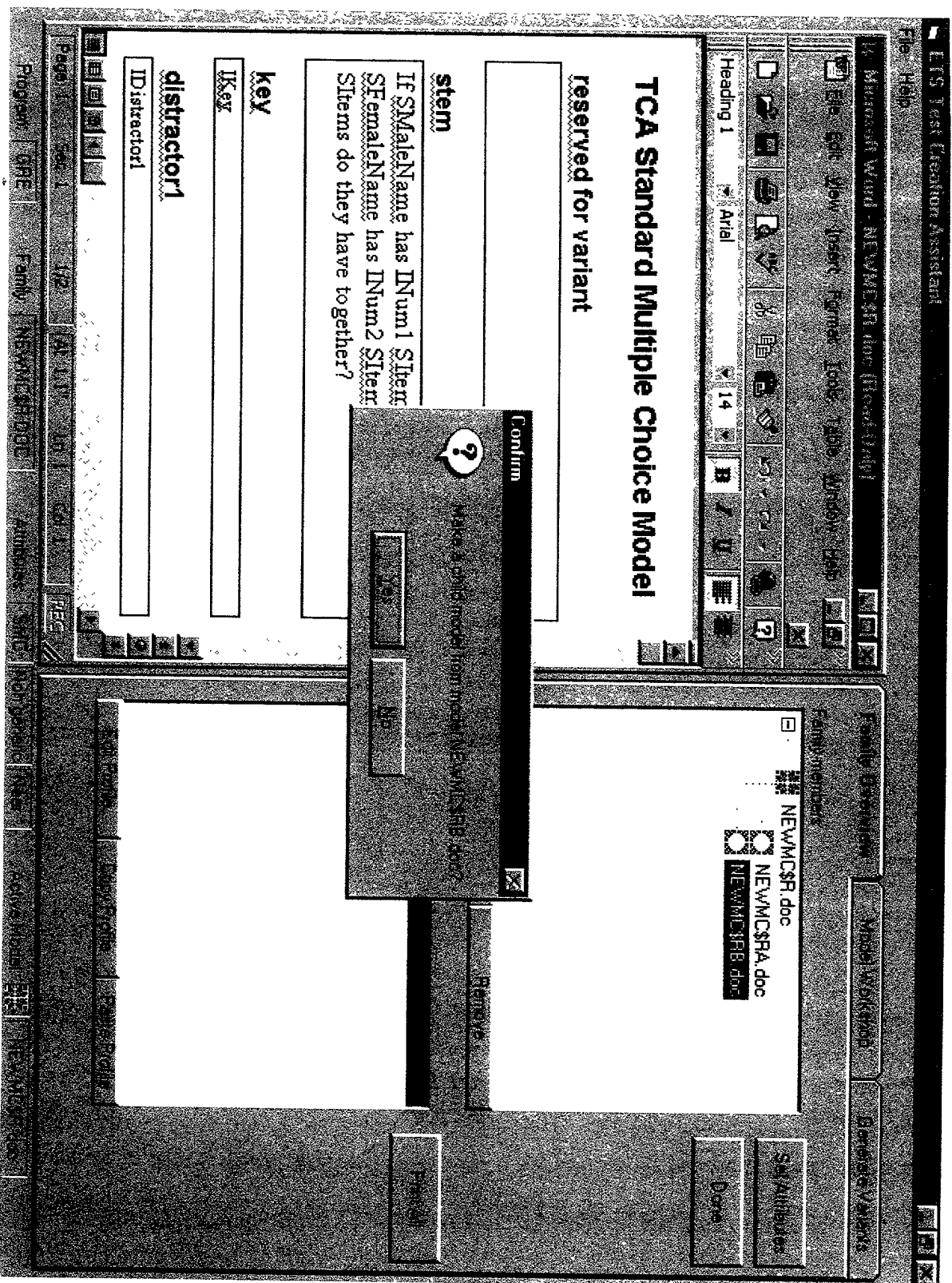


FIG. 74

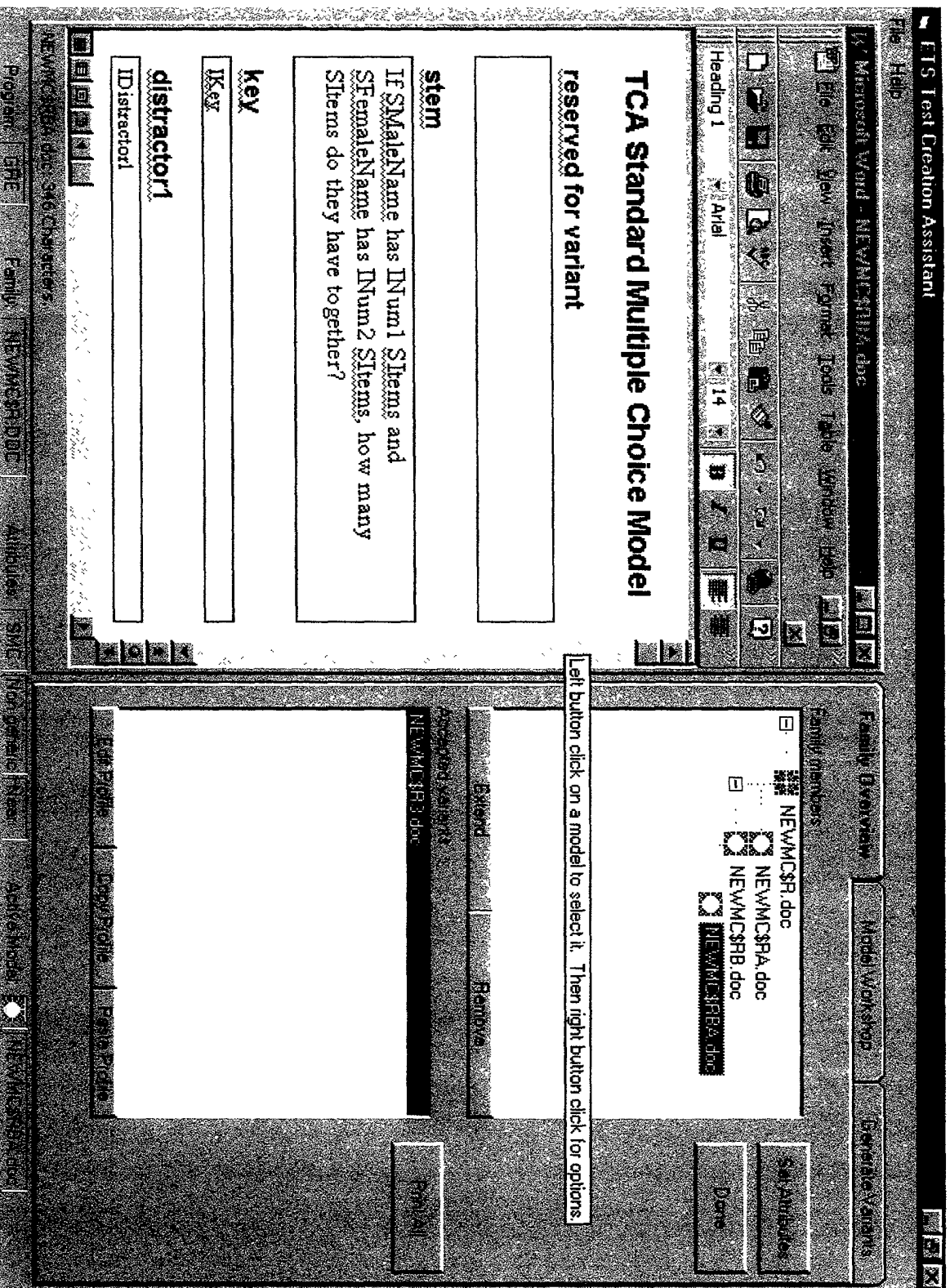


FIG. 750100

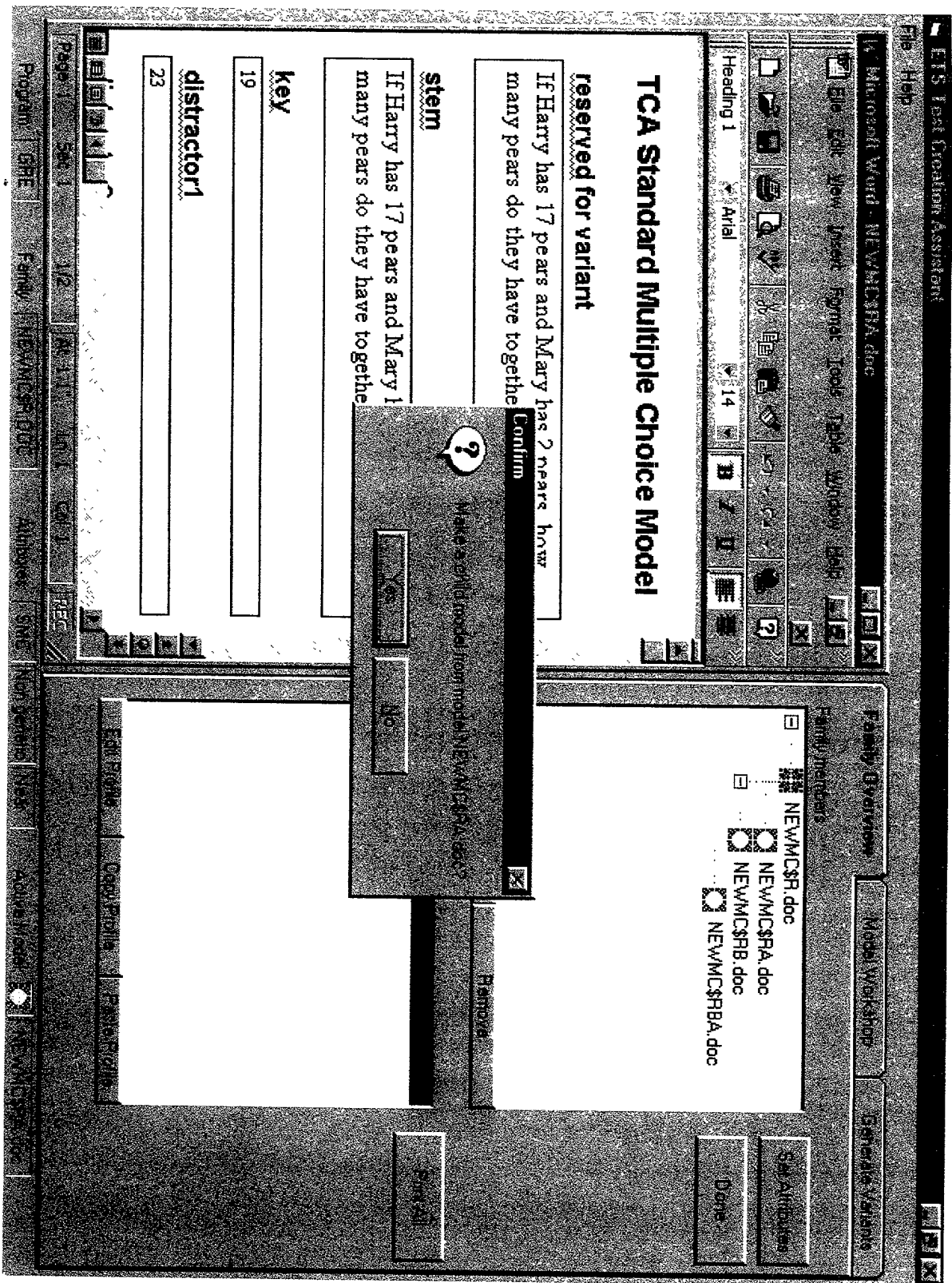
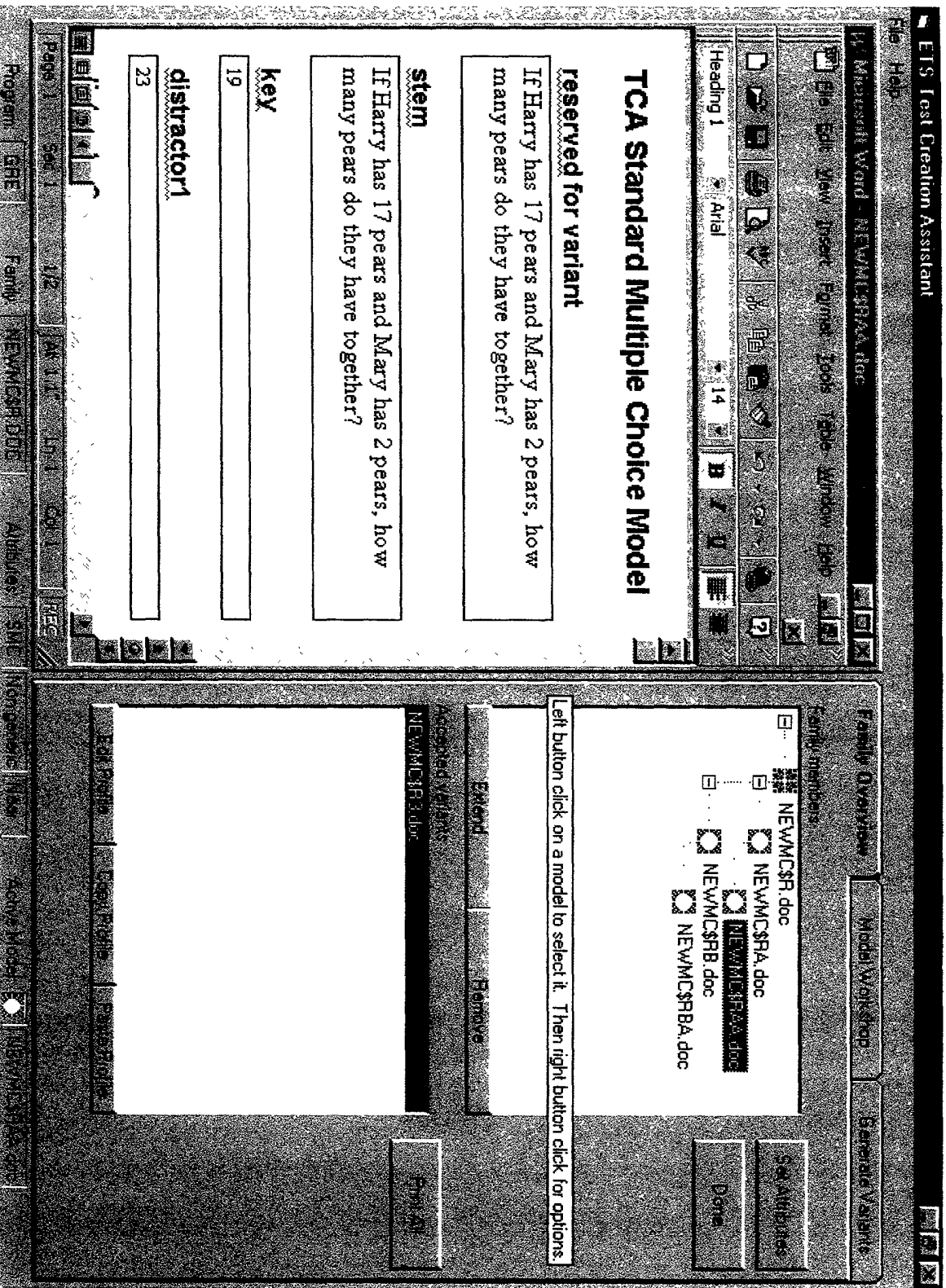


FIG. 76



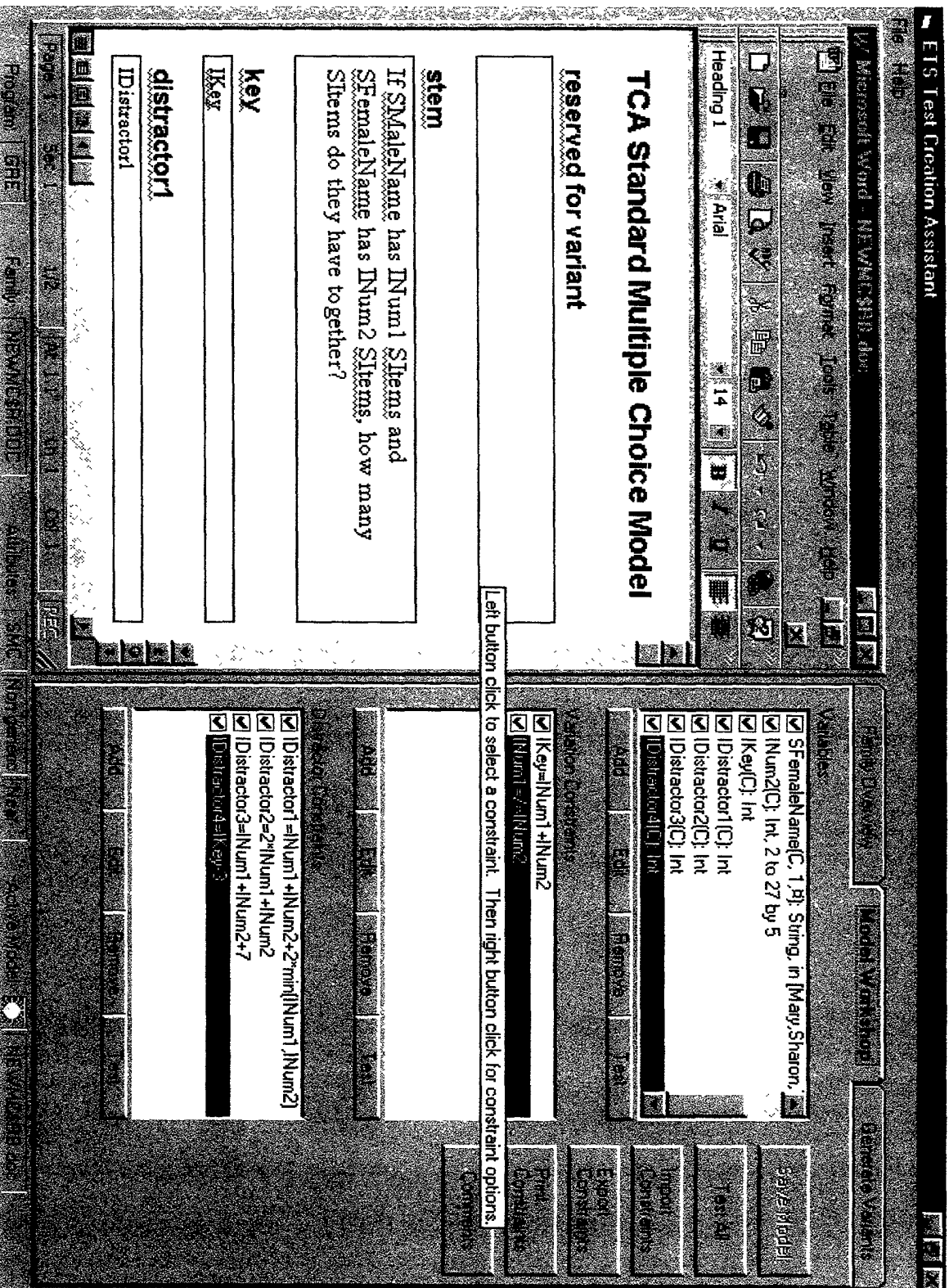


FIG. 78

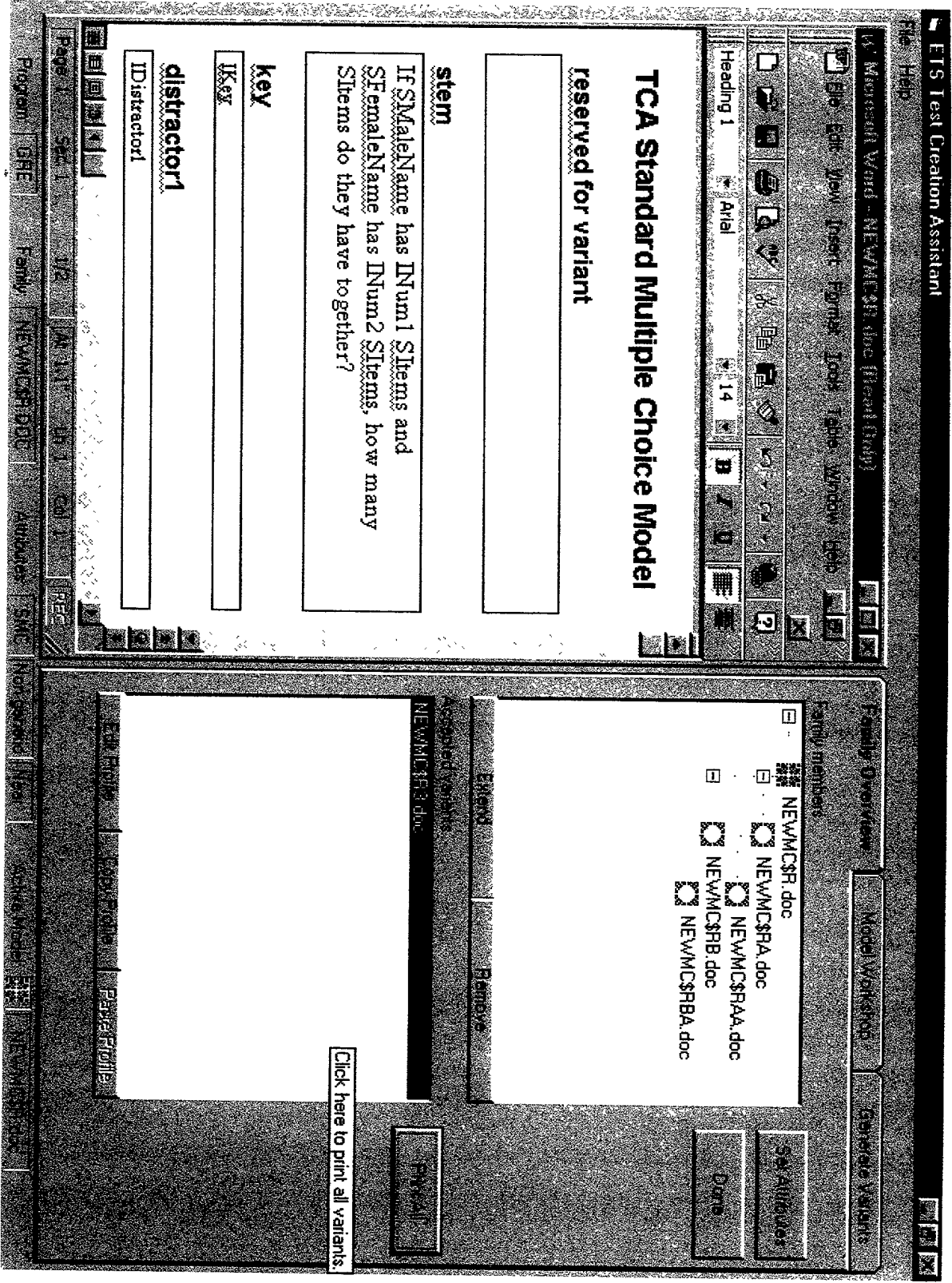


FIG. 79

Variables and constraints for model NEWMC\$R

Variables:

Variable name: SMaleName

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

John

Tom

Richard

Michael

Steve

Phil

Jeff

Peter

Harry

Variable name: INum1

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 2 to 26 by 3

Variable name: SItems

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

apples

oranges

pears

marbles

pennies

comic books

pieces of bubble gum

pencils

crayons

Variable name: SFemaleName

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

Mary

Sharon

Tina

Michelle

Variables and constraints for model NEWMC\$R

Susan

Linda

Crystal

Deidre

Variable name: INum2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 2 to 27 by 5

Variable name: IKey

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor1

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor3

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor4

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Constraints:

Variation constraints:

Constraint: $IKey = INum1 + INum2$

Status: Enabled

Constraint: $INum1 \neq INum2$

Status: Enabled

Distractor constraints:

Constraint: $IDistractor1 = INum1 + INum2 + 2 * \min(INum1, INum2)$

Status: Enabled

Constraint: $IDistractor2 = 2 * INum1 + INum2$

Status: Enabled

Constraint: $IDistractor3 = INum1 + INum2 + 7$

Variables and constraints for model NEWMC\$RA

Variables:

Variable name: SMaleName

Type: String
Status: Enabled
Checksum: Enabled
Indexed: False

Values:

- John
- Tom
- Richard
- Michael
- Steve
- Phil
- Jeff
- Peter
- Harry

Variable name: INum1

Type: Integer
Status: Enabled
Checksum: Enabled
Is independent = True, Range: from 2 to 26 by 3

Variable name: SItems

Type: String
Status: Enabled
Checksum: Enabled
Indexed: False

Values:

- apples
- oranges
- pears
- marbles
- pennies
- comic books
- pieces of bubble gum
- pencils
- crayons

Variable name: SFemaleName

Type: String
Status: Enabled
Checksum: Enabled
Indexed: False

Values:

- Mary
- Sharon
- Tina
- Michelle

FIG.82A

Variables and constraints for model NEWMC\$RA

Susan

Linda

Crystal

Deidre

Variable name: INum2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 2 to 27 by 5

Variable name: IKey

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor1

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor3

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor4

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Constraints:

Variation constraints:

Constraint: $IKey = INum1 + INum2$

Status: Enabled

Constraint: $INum1 \neq INum2$

Status: Enabled

Distractor constraints:

Constraint: $IDistractor1 = INum1 + INum2 + 2 * \min(INum1, INum2)$

Status: Enabled

Constraint: $IDistractor2 = 2 * INum1 + INum2$

Status: Enabled

Constraint: $IDistractor3 = INum1 + INum2 + 7$

Variables and constraints for model NEWMC\$RA

Status: Enabled

Constraint: IDistractor4=IKey-3

Status: Enabled

[illegible]

FILE: NEWMC\$R.doc

TCA Standard Multiple Choice Model

reserved for variant

stem

If SMaleName has INum1 SItems and
SFemaleName has INum2 SItems, how many
SItems do they have together?

key

IKey

distractor1

IDistractor1

distractor2

IDistractor2

distractor3

IDistractor3

distractor4

IDistractor4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 83

FILE: NEWMC\$R3.doc

TCA Standard Multiple Choice Model

reserved for variant

If Tom has 2 comic books and Crystal has 12 comic books, how many comic books do they have together?

stem

If Tom has 2 comic books and Crystal has 12 comic books, how many comic books do they have together?

key

14

distractor1

18

distractor2

16

distractor3

21

distractor4

11

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 84

TCA Standard Multiple Choice Model

reserved for variant

If Harry has 17 pears and Mary has 2 pears, how many pears do they have together?

stem

If Harry has 17 pears and Mary has 2 pears, how many pears do they have together?

key

19

distractor1

23

distractor2

36

distractor3

26

distractor4

16

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 85

TCA Standard Multiple Choice Model

reserved for variant

If Harry has 17 pears and Mary has 2 pears, how many pears do they have together?

stem

If Harry has 17 pears and Mary has 2 pears, how many pears do they have together?

key

19

distractor1

23

distractor2

36

distractor3

26

distractor4

16

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 86

FILE: NEWMC\$RB.doc

TCA Standard Multiple Choice Model

reserved for variant

stem

If SMaleName has INum1 SItems and
SFemaleName has INum2 SItems, how many
SItems do they have together?

key

IKey

distractor1

IDistractor1

distractor2

IDistractor2

distractor3

IDistractor3

distractor4

IDistractor4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 87

FILE: NEWMC\$RBA.doc

TCA Standard Multiple Choice Model

reserved for variant

stem

If SMaleName has INum1 SItems and
SFemaleName has INum2 SItems, how many
SItems do they have together?

key

IKey

distractor1

IDistractor1

distractor2

IDistractor2

distractor3

IDistractor3

distractor4

IDistractor4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 88

Microsoft Word 6.03 Help
File Edit View Insert Format Tools Table Window Help
New Open Recent Undo Redo Cut Copy Paste Find Replace
Bold Italic Underline Bulleted List Numbered List
Decrease Indent Increase Indent Link Unlink Page Setup
Print Screen Window Help

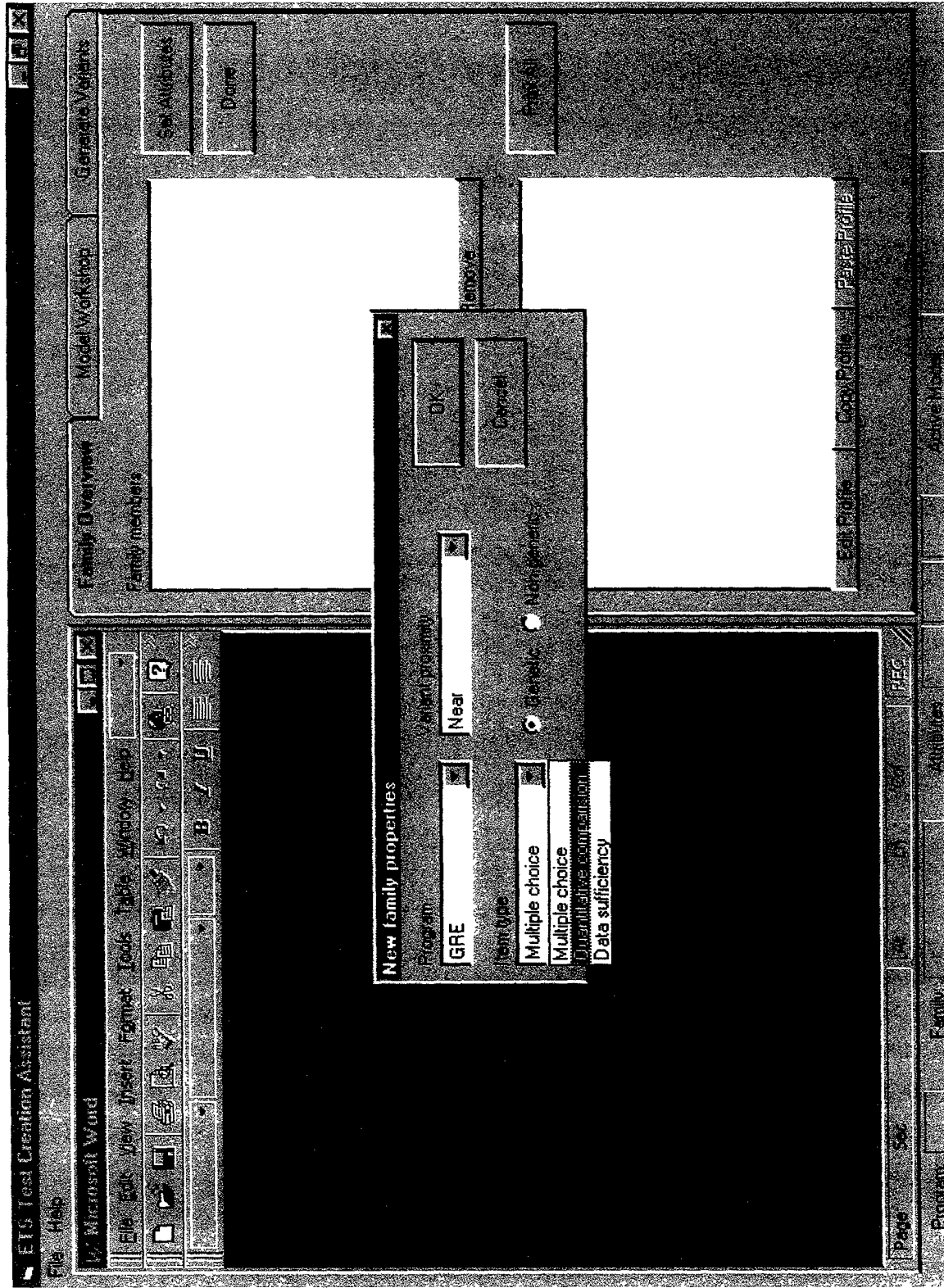


FIG. 89

Microsoft Word - NEWQCSR.doc

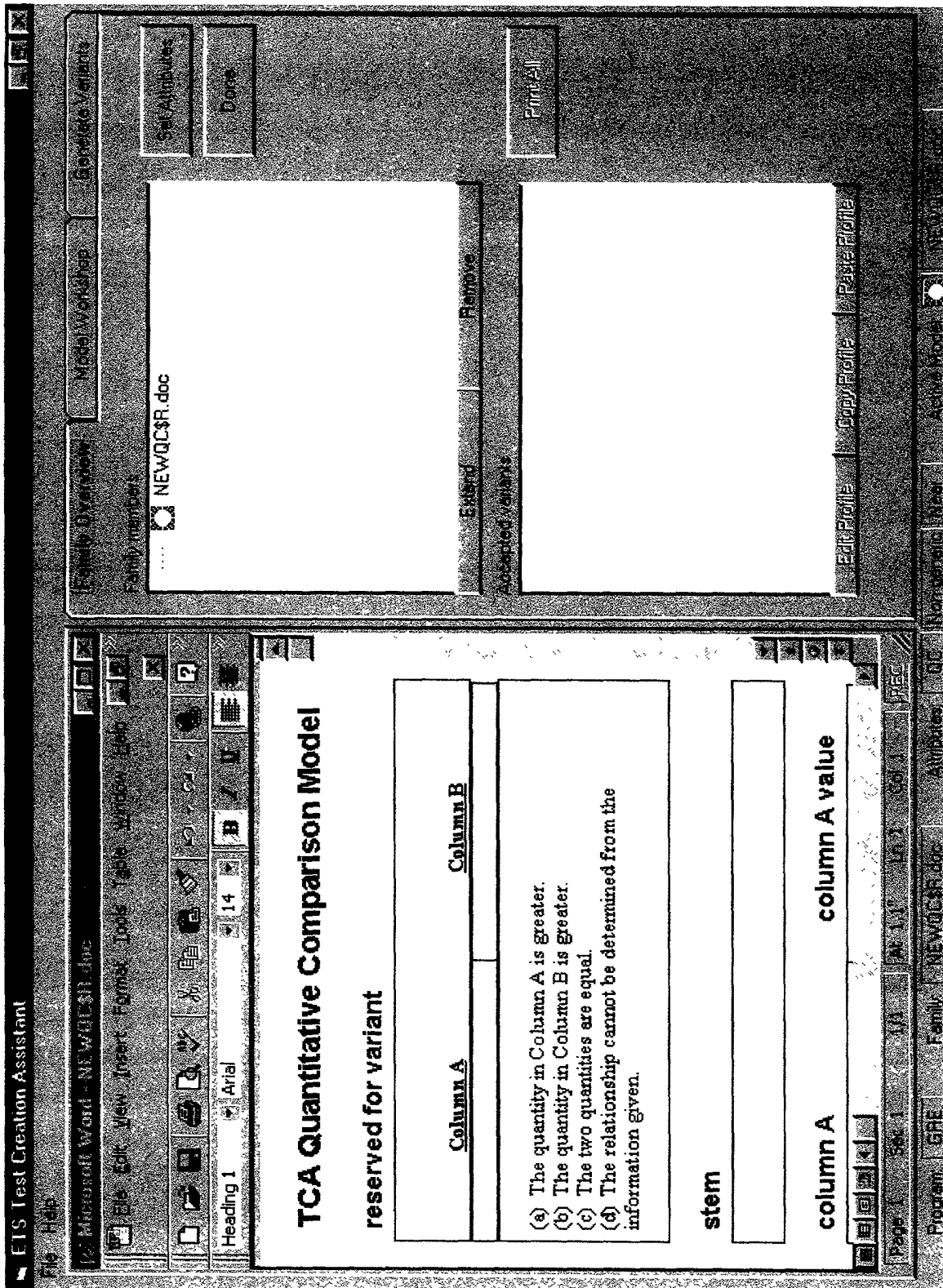


FIG. 90

TCA Quantitative Comparison Model

reserved for variant

<u>Column A</u>	<u>Column B</u>
(a) The quantity in Column A is greater. (b) The quantity in Column B is greater. (c) The two quantities are equal. (d) The relationship cannot be determined from the information given.	

stem

--

column A column A value

--	--

column B column B value

--	--

key

Key

scratch pad

Scratch Pad Area

FIG. 91

Microsoft Word - NEWDSR.doc

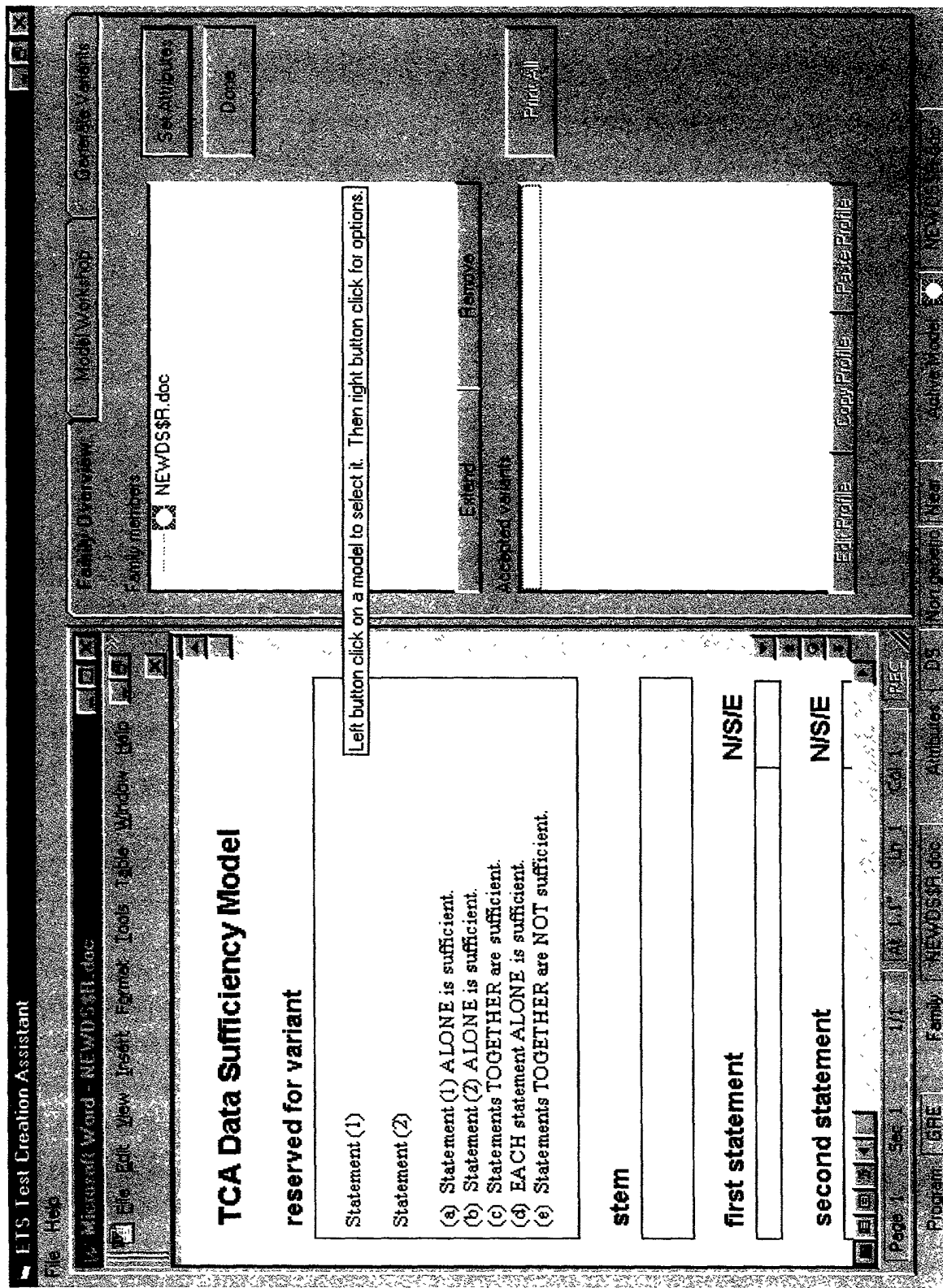


FIG. 92

TCA Data Sufficiency Model

reserved for variant

Statement (1)

Statement (2)

- (a) Statement (1) ALONE is sufficient.
- (b) Statement (2) ALONE is sufficient.
- (c) Statements TOGETHER are sufficient.
- (d) EACH statement ALONE is sufficient.
- (e) Statements TOGETHER are NOT sufficient.

stem

first statement

N/S/E

second statement

N/S/E

key

Key

scratch pad

Scratch
Pad
Area

FIG. 93

Microsoft Word - MICNEWMSCH.doc (Read-Only)
File Edit View Insert Format Tools Table Window Help

EIS Test Creation Assistant

File Help

Microsoft Word - MICNEWMSCH.doc (Read-Only)

File Edit View Insert Format Tools Table Window Help

TCA Standard Multiple Choice Model

reserved for variant

stem

If SMaleName had INum1 SThing and SFemaleName had INum2 SThing how many SThing did they have together?

key

Key

Page: 1 Sec: 1 1/2 1st 1st Col: 1 REC

Program GRE Family MICNEWMSCH.DOC Attributes SMC Not Shared New Version Model

Form Overview

Model Workshop

Generate Variants

Save Model

Tab All

Import Constraints

Export Constraints

Run Constraints

Commit

Variables

☒ SFemaleName(C, 1, 8): String, in [Holly, Mary, Te

☒ INum2(C): Int, 4 to 12 by 1

☒ IKey(C): Int

☒ IDistractor1(C): Int

☒ IDistractor2(C): Int

☒ IDistractor3(C): Int

☒ IDistractor4(C): Int

Add Browse Remove

Version Constraints

☒ IKey = INum1 + INum2

Add Browse Remove

Distractor Constraints

☒ IDistractor1 = INum1 - INum2

☒ IDistractor2 = INum1 * INum2

☒ IDistractor3 = IDistractor1 + IDistractor2

☒ IDistractor4 = 2 * INum1

Add Browse Remove

FIG. 94

TCA Standard Multiple Choice Model

reserved for variant

If Bill had 2 apples and Teresa had 5 apples, how many apples did they have together?

- A. 3
 - B. 4
 - C. 7
 - D. 10
 - E. 13
- Key is C

stem

If Bill had 2 apples and Teresa had 5 apples, how many apples did they have together?

key

7

distractor1

3

distractor2

10

distractor3

13

distractor4

4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 95

TCA Standard Multiple Choice Model

reserved for variant

If Bill had 2 apples and Joan had 4 apples, how many apples did they have together?

- A. 2
- B. 4
- C. 6
- D. 8
- E. 10

Key is C

stem

If Bill had 2 apples and Joan had 4 apples, how many apples did they have together?

key

6

distractor1

2

distractor2

8

distractor3

10

distractor4

4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 96

TCA Standard Multiple Choice Model

reserved for variant

If Bill had 2 apples and Joan had 4 apples, how many apples did they have together?

- A. 2
 - B. 4
 - C. 6
 - D. 8
 - E. 10
- Key is C

stem

If Bill had 2 apples and Joan had 4 apples, how many apples did they have together?

key

6

distractor1

2

distractor2

8

distractor3

10

distractor4

4

distractor5

Distractor5

distractor6

Distractor6

distractor7

Distractor7

distractor8

Distractor8

scratch pad

Scratch Pad Area

FIG. 97

Variables and constraints for model MICNEWMC\$R

Variables:

Variable name: SMaleName

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

Michael

Bill

Harry

Roger

Variable name: INum1

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 2 to 8 by 1

Variable name: SThing

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

apples

uzis

Variable name: SFemaleName

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

Holly

Mary

Teresa

Joan

Variable name: INum2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 4 to 12 by 1

Variable name: IKey

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor1

Type: Integer

Variables and constraints for model MICNEWMC\$R

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor3

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: IDistractor4

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Constraints:

Variation constraints:

Constraint: IKey = INum1 + INum2

Status: Enabled

Distractor constraints:

Constraint: IDistractor1 = |INum1 - INum2|

Status: Enabled

Constraint: IDistractor2 = INum1 * INum2

Status: Enabled

Constraint: IDistractor3 = IDistractor1 + IDistractor2

Status: Enabled

Constraint: IDistractor4 = 2 * INum1

Status: Enabled

Microsoft Word - MICNEWQCR.doc (Read-Only)
File Edit View Insert Format Tools Table Window Help

EIS Test Creation Assistant

File Help

Microsoft Word - MICNEWQCR.doc (Read-Only)

File Edit View Insert Format Tools Table Window Help

TCA Quantitative Comparison Model

reserved for variant

Column A	Column B
(a) The quantity in Column A is greater.	
(b) The quantity in Column B is greater.	
(c) The two quantities are equal.	
(d) The relationship cannot be determined from the information given.	

stem

An article of clothing was reduced in price by x percent from \$OriginalPrice to \$ReducedPrice. Later, the price was increased y percent to return the price to \$OriginalPrice.

column A	column B
x	y
	1

MICNEWQCR.doc 432 Characters (Read-Only)

Program GRE Family MICNEWQCR.doc Attributes Q1

Family Overview Model Workspace Generate Variants

Family members

MICNEWQCR.doc

MICNEWQCR\$RA.doc

Set Attributes

Done

Extend Remove

Accepted Variants

MICNEWQCR\$R5.doc

MICNEWQCR\$R6.doc

MICNEWQCR\$R7.doc

MICNEWQCR\$R8.doc

MICNEWQCR\$R9.doc

MICNEWQCR\$R8.doc

Set Profile Copy Profile Paste Profile

FIG. 99

TCA Quantitative Comparison Model

reserved for variant

<u>Column A</u>	<u>Column B</u>
<p>(a) The quantity in Column A is greater. (b) The quantity in Column B is greater. (c) The two quantities are equal. (d) The relationship cannot be determined from the information given.</p>	

stem

An article of clothing was reduced in price by x percent from \$IOriginalPrice to \$IReducedPrice. Later, the price was increased by y percent to return the price to \$IOriginalPrice.

column A

column B

x	y
x + 1	y - 1

key

Key

scratch pad

Scratch
Pad
Area

FIG. 100

TCA Quantitative Comparison Model

reserved for variant

An article of clothing was reduced in price by x percent from \$20 to \$16. Later, the price was increased by y percent to return the price to \$20.

<u>Column A</u>	<u>Column B</u>
$x + 1$	$y - 1$
<p>(a) The quantity in Column A is greater. (b) The quantity in Column B is greater. (c) The two quantities are equal. (d) The relationship cannot be determined from the information given.</p>	

stem

An article of clothing was reduced in price by x percent from \$20 to \$16. Later, the price was increased by y percent to return the price to \$20.

column A	column B
x	y
$x + 1$	$y - 1$

key

Key

scratch pad

Scratch
Pad
Area

FIG. 101

TCA Quantitative Comparison Model

reserved for variant

An article of clothing was reduced in price by x percent from \$25 to \$20. Later, the price was increased by y percent to return the price to \$25.

<u>Column A</u>	<u>Column B</u>
$x + 1$	y
<p>(a) The quantity in Column A is greater. (b) The quantity in Column B is greater. (c) The two quantities are equal. (d) The relationship cannot be determined from the information given.</p>	

stem

An article of clothing was reduced in price by x percent from \$25 to \$20. Later, the price was increased by y percent to return the price to \$25.

column A	column B
x	y
$x + 1$	$y - 1$

key

Key

scratch pad

Scratch
Pad
Area

FIG. 102

Microsoft Word - MICNEWDS\$R.doc

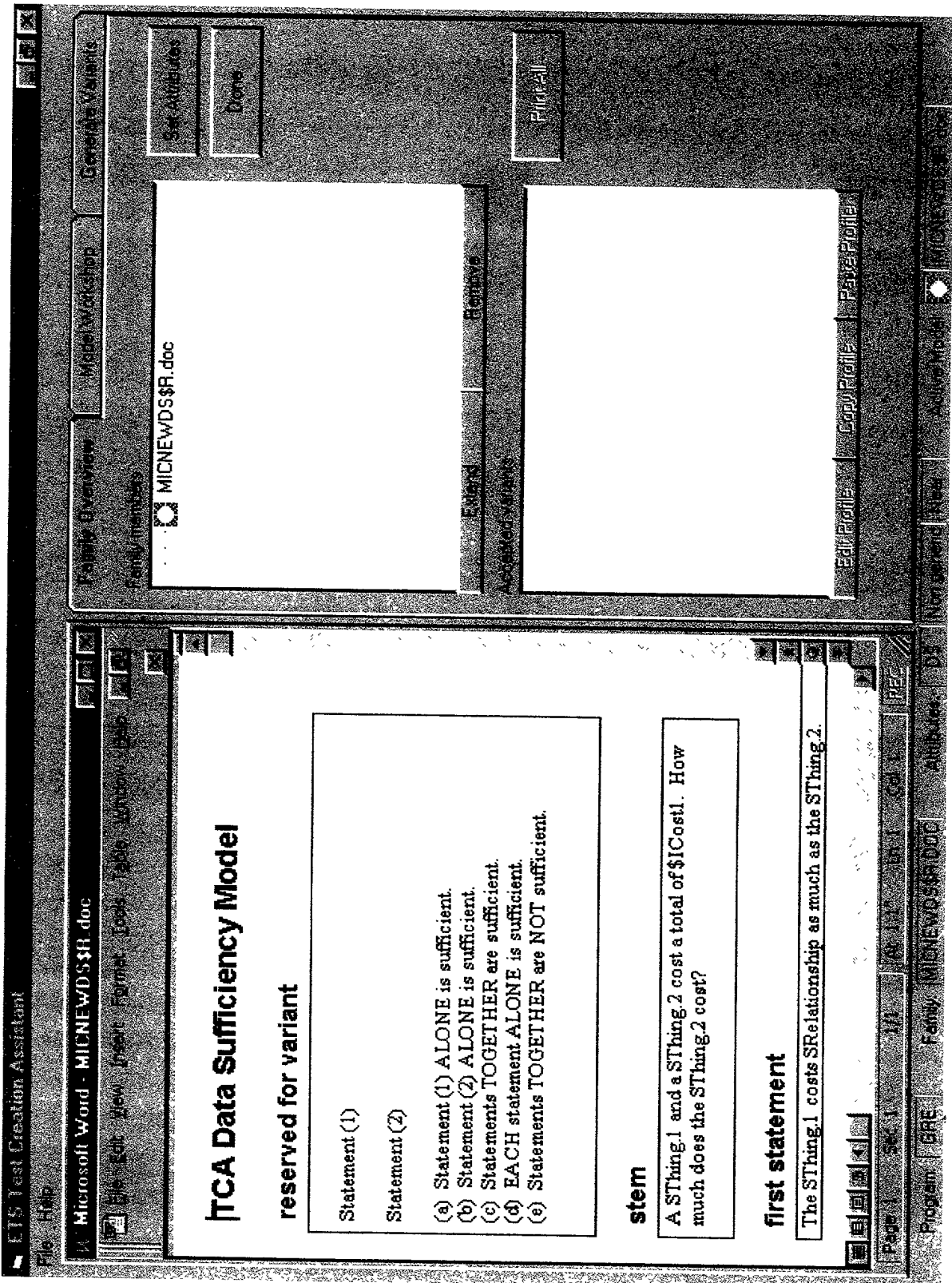


FIG. 103

Microsoft Word - MIENEWD578.DOC

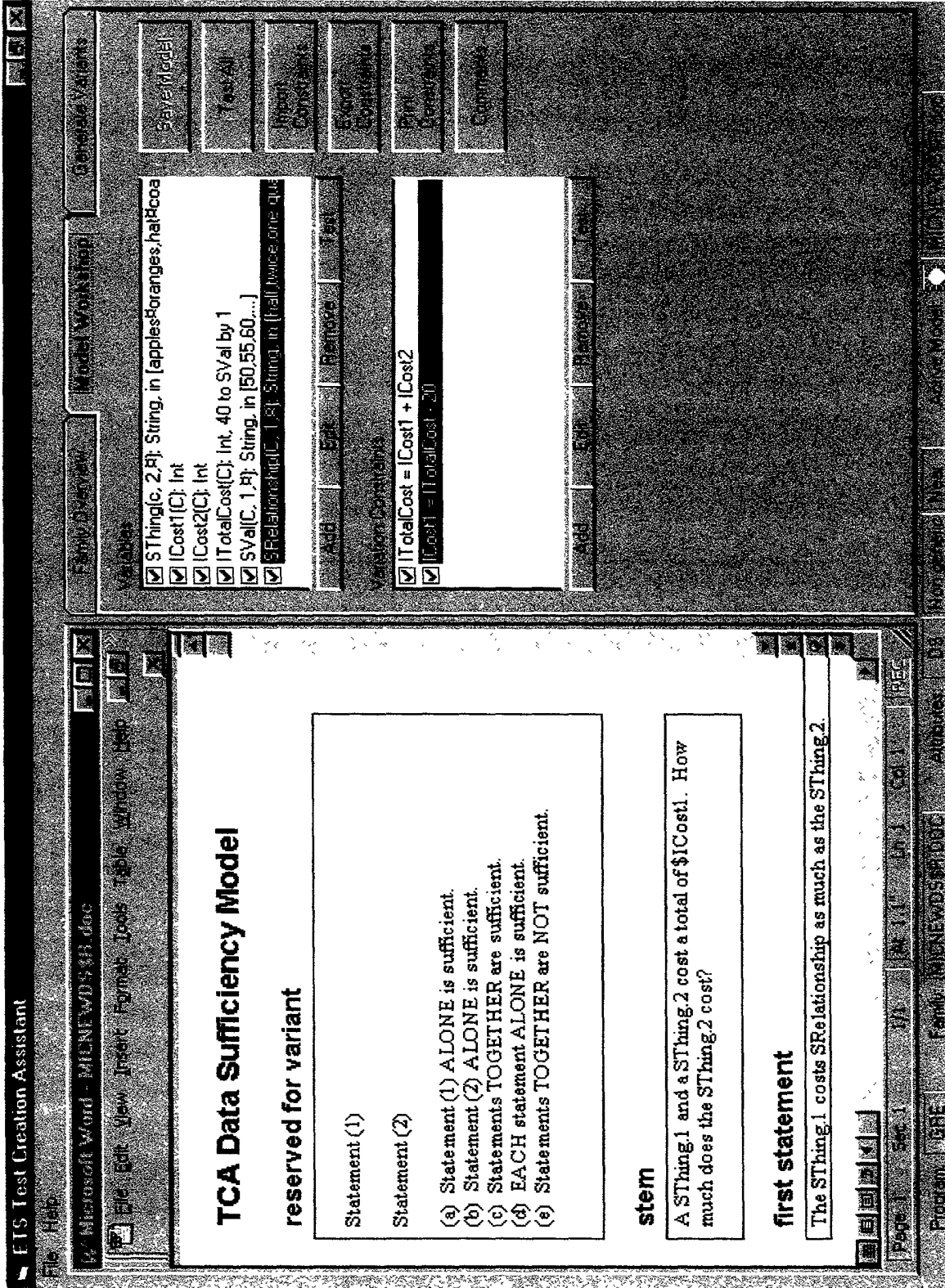


FIG. 104

TCA Data Sufficiency Model

reserved for variant

Statement (1)

Statement (2)

- (a) Statement (1) ALONE is sufficient.
- (b) Statement (2) ALONE is sufficient.
- (c) Statements TOGETHER are sufficient.
- (d) EACH statement ALONE is sufficient.
- (e) Statements TOGETHER are NOT sufficient.

stem

A SThing.1 and a SThing.2 cost a total of \$ICost1. How much does the SThing.2 cost?

first statement

The SThing.1 costs SRelationship as much as the SThing.2.

second statement

The SThing.1 costs \$ICost2.

key

Key

scratch pad

Scratch
Pad
Area

FIG. 105

Variables and constraints for model MICNEWDS\$R

Variables:

Variable name: SThing

Type: String

Status: Enabled

Checksum: Disabled

Indexed: True

Value Sets:

Values:

1. apples

2. oranges

Values:

1. hat

2. coat

Variable name: ICost1

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: ICost2

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = False

Variable name: ITotalCost

Type: Integer

Status: Enabled

Checksum: Enabled

Is independent = True, Range: from 40 to SVal by 1

Variable name: SVal

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

50

55

60

65

Variable name: SRelationship

Type: String

Status: Enabled

Checksum: Enabled

Indexed: False

Values:

half

twice

one quarter
three times

FIG. 107 is a block diagram of a system architecture for a high-level constraint solver.

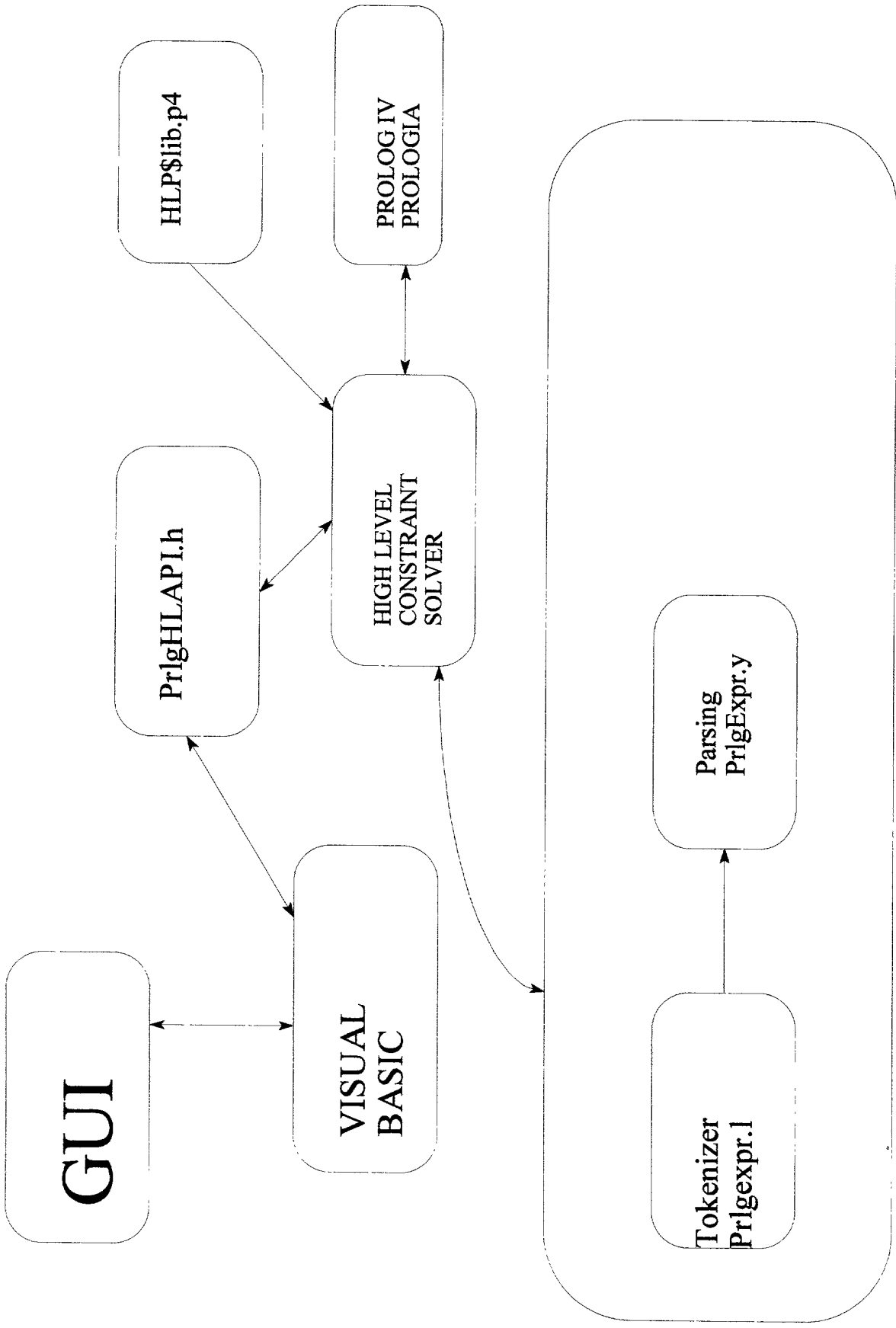


FIG. 107

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT EXAMINING OPERATION

ATTN'Y DOCKET NO.: ETS-TCA

APPLICATION OF: PETER BRITTINGHAM, MARY E. MORLEY, MARK K.
SINGLEY, MARK G. ZELMAN, KRISHNA N. JHA,
JAMES H. FIFE, ROBERT L. RARICH, IRVIN R.
KATZ, RANDY E. BENNETT

FOR: COMPUTER-BASED TEST-ITEM GENERATION AND
CLONING

VISUAL BASIC SOURCE CODE APPENDIX
(VBSCA 1-469)

VISUAL BASIC SOURCE CODE APPENDIX TABLE OF CONTENTS¹

	TCA.vbp	VBSCA -1-
5	AXProlog.vbp	VBSCA -4-
	Common.bas	VBSCA -5-
	Main.bas	VBSCA -6-
	modUtil.bas	VBSCA -7-
	MTAPI.BAS	VBSCA -12-
10	MTDeclaration.bas	VBSCA -17-
	MTUtil.bas	VBSCA -21-
	Timer.bas	VBSCA -28-
	Contraint.frm	VBSCA -29-
	EditConstraint.frm	VBSCA -50-
	Form1.frm	VBSCA -52-
	frmAbout.frm	VBSCA -54-
	frmAttributes.frm	VBSCA -55-
	frmComments.frm	VBSCA -60-
	frmDifficulty.frm	VBSCA -62-
20	frmDrag.frm	VBSCA -76-
	frmIED.frm	VBSCA -79-
	frmIndexedString.frm	VBSCA -81-

¹ All software COPYRIGHT 1999 ETS except for MTAPI.BAS

5
10
15
20

frmNew.frm	VBSCA -89-
frmNewModel.frm	VBSCA -94-
frmProgram.frm	VBSCA -97-
frmProgress.frm	VBSCA -100-
frmProlog.frm	VBSCA -102-
frmSplash.frm	VBSCA -104-
SetPrecision.frm	VBSCA -108-
String.frm	VBSCA -111-
TCA.FRM	VBSCA -114-
Variable.frm	VBSCA -217-
Application.cls	VBSCA -254-
CClones.cls	VBSCA -255-
CConstraints.cls	VBSCA -261-
Checksum.cls	VBSCA -268-
Clone.cls	VBSCA -270-
CModels.cls	VBSCA -279-
Constraint.cls	VBSCA -283-
ConstraintSolver.cls	VBSCA -288-
CVariables.cls	VBSCA -297-
CVariants.cls	VBSCA -308-
DifficultyEstimate.cls	VBSCA -311-
DocStatus.cls	VBSCA -313-
DSMODEL.CLS	VBSCA -314-

Family.cls	VBSCA -322-
File.cls	VBSCA -328-
FileFind.cls	VBSCA -333-
GMATDifficultyEstimate.cls	VBSCA -336-
GREDifficultyEstimate.cls	VBSCA -340-
IniFile.cls	VBSCA -345-
LockedItem.cls	VBSCA -350-
Model.cls	VBSCA -362-
PrintModel.cls	VBSCA -381-
Progress.cls	VBSCA -384-
Prolog.cls	VBSCA -386-
PSMODEL.cls	VBSCA -392-
QCModel.cls	VBSCA -403-
StringSolver.cls	VBSCA -410-
StringSolverx.cls	VBSCA -412-
SubString.cls	VBSCA -413-
Value.cls	VBSCA -417-
VarFraction.cls	VBSCA -419-
Variable.cls	VBSCA -428-
VarInteger.cls	VBSCA -432-
VarReal.cls	VBSCA -439-
VarString.cls	VBSCA -449-
VarUntyped.cls	VBSCA -456-

Win32API.cls	VBSCA -462-
Word.cls	VBSCA -466-

```

' TCA.vbp
Type=Exe
Reference=*\\G{00020430-0000-0000-C000-000000000046}#2.0#0#...\\WINNT\\System32\\
StdOle2.Tlb#OLE Automation
5 Reference=*\\G{00020905-0000-0000-C000-000000000046}#8.0#409#...\\Microsoft
Office\\Office\\MSWORD8.OLB#Microsoft Word 8.0 Object Library
Reference=*\\G{953298D7-F0DE-11D2-AED3-000000000000}#13.0#0#AXProlog.exe#AXProlog
10 Object={FE0065C0-1B7B-11CF-9D53-00AA003C9CB6}#1.1#0; COMCT232.OCX
Object={6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0; COMCTL32.OCX
Object={BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0; TABCTL32.OCX
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0; COMDLG32.OCX
Form=TCA.frm
Module=Util; modUtil.bas
15 Class=Model; Model.cls
Class=Constraint; Constraint.cls
Class=Variable; Variable.cls
Class=TCAApplication; Application.cls
Module=Startup; Main.bas
20 Form=Variable.frm
Class=CVariables; CVariables.cls
Class=CConstraints; CConstraints.cls
Form=Constraint.frm
Class=MSWord; Word.cls
25 Form=frmSplash.frm
Class=VarInteger; VarInteger.cls
Class=VarReal; VarReal.cls
Class=VarFraction; VarFraction.cls
Class=VarString; VarString.cls
30 Form=frmIndexedString.frm
Class=File; File.cls
Class=CClones; CClones.cls
Class=IniFile; IniFile.cls
Class=Win32API; Win32API.cls
35 Class=CModels; CModels.cls
Class=Clone; Clone.cls
Form=frmAttributes.frm
Class=Family; Family.cls
Class=DocStatus; DocStatus.cls
40 Class=Checksum; Checksum.cls
Form=frmProgress.frm
Class=Progress; Progress.cls
Form=frmDifficulty.frm
Class=DifficultyEstimate; DifficultyEstimate.cls

```

```

Class=GREDDifficultyEstimate; GREDDifficultyEstimate.cls
Class=SMCModel; PSModel.cls
Class=QCModel; qcmodel.cls
Class=DSModel; dsmodel.cls
5 Class=VarUntyped; VarUntyped.cls
Class=LockedItem; LockedItem.cls
Class=GMATDDifficultyEstimate; GMATDDifficultyEstimate.cls
Form=frmAbout.frm
Form=frmNew.frm
10 Form=String.frm
Class=SubString; SubString.cls
Class=ConstraintSolver; ConstraintSolver.cls
Class=StringSolver; StringSolver.cls
Class=Value; Value.cls
15 Class=PrintModel; PrintModel.cls
Module=MTAPI; MTAPI.bas
Module=MTDeclarations; MTDeclarations.bas
Module=MTUtil; MTUtil.bas
Form=frmProlog.frm
20 ResFile32="Tca.res"
IconForm="frmTCA"
Startup="Sub Main"
HelpFile=""
Title="TCA"
25 ExeName32="TCA.exe"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0"
30 MajorVer=0
MinorVer=1
RevisionVer=145
AutoIncrementVer=1
ServerSupportFiles=0
35 VersionCompanyName="ETS"
CompilationType=0
OptimizationType=2
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
40 NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FIPointCheck=0
FDIVCheck=0
45 UnroundedFP=0

```

MaxNumberOfThreads=1

[illegible]


```

' AXProlog.vbp
Type=OleExe
Reference=*G{00020430-0000-0000-C000-0000000000046}#2.0#0#...\WINNT\System32\
STDOLE2.TLB#OLE Automation
5 Reference=*G{3D5C6BF0-69A3-11D0-B393-00A0C9055D8E}#1.0#0#...\Common
Files\designer\MSDERUN.DLL#Microsoft Data Environment Instance 1.0
Reference=*G{00000200-0000-0010-8000-00AA006D2EA4}#2.0#0#...\Common
Files\system\ado\msado20.tlb#Microsoft ActiveX Data Objects 2.0 Library
Class=Prolog; Prolog.cls
10 Module=Module1; Timer.bas
Class=File; File.cls
Startup="(None)"
HelpFile=""
ExeName32="AXProlog.exe"
15 Command32=""
Name="AXProlog"
HelpContextID="0"
CompatibleMode="1"
CompatibleEXE32="AXProlog.exe"
20 MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
25 VersionCompanyName="ETS"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
30 NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
35 UnroundedFP=0
StartMode=1
Unattended=-1
Retained=0
ThreadPerObject=-1
40 MaxNumberOfThreads=1
DebugStartupOption=0

```

' Common.bas
Attribute VB_Name = "Common"

VBSCA -5-

' Main.bas

Attribute VB_Name = "StartUp"

Option Explicit

Public Const READ_UNTIL_EOF = 0

5 Public Const INI_DIRECTORY = "C:\TCS\TCA\OUT\TCAOUT.INI"

Public Const IN_DIRECTORY = "C:\TCS\TCA\IN\"

Public Const OUT_DIRECTORY = "C:\TCS\TCA\OUT\"

Public Const LOCKED_ITEM_NAME = "TCATEMP.DOC"

Public Const LVM_FIRST = &H1000

10 Public Const LVM_SETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 54

Public Const LVM_GETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 55

Public Const LVS_EX_FULLROWSELECT = &H20

Public Const HALT_FN = "C:\HALT.TCA"

Public Const STRING_DELIMITER = 164

15 Private Sub Main()

Dim MyApp As New TCAApplication

If App.PreviousInstance Then

20 Call MsgBox("Only one instance of TCA may be run at a time!", _
vbExclamation, "Error")

Exit Sub

End If

' 10 seconds for component timeout

25 App.OleRequestPendingTimeout = 10000

MyApp.Run

End Sub

```

' modUtil.bas
Attribute VB_Name = "Util"
Option Explicit

' Capitalizes the first letter of a string if it's a lower case letter
5 Sub CapitalizeString(strInput As String)

    Dim str1, str2 As String
    Dim intStrLen As Integer

    intStrLen = Len(strInput)

    If (intStrLen > 0) Then
10         str1 = UCase(left(strInput, 1))
        End If

    If (intStrLen > 1) Then
        str2 = right(strInput, intStrLen - 1)
    End If

15     strInput = str1 & str2

End Sub

' Selects contents of text box for easy editing
Sub txtSelectAll(txtTextBox As TextBox)

    ' Automatically select all text
20     txtTextBox.SelStart = 0
    txtTextBox.SelLength = Len(txtTextBox.Text)

End Sub

' Checks to see if a file exists

Function FileExists(ByVal strFN As String) As Boolean

25     Dim intFNum As Integer

    ' Get the file number
    intFNum = FreeFile

    ' Open the file and trap any errors
    On Error GoTo NotFound

```

```
Open strFN For Binary Access Read As #intFNum
On Error GoTo 0
```

```
Close #intFNum
```

```
FileExists = True
Exit Function
```

```
NotFound:
```

```
' Close the file
Close #intFNum
FileExists = False
Exit Function
```

```
End Function
```

```
' extracts the path from a path/filename string
```

```
Function ExtractPath(ByVal strFN As String) As String
```

```
Dim varI1 As Variant
Dim varI2 As Variant
```

```
' find the last "\" in the string
varI1 = 0
```

```
Do
```

```
    varI2 = varI1
```

```
    varI1 = InStr(varI2 + 1, strFN, "\" )
```

```
Loop Until varI1 = 0
```

```
ExtractPath = Mid(strFN, 1, varI2)
```

```
End Function
```

```
' extracts the file name from a path/filename string
```

```
Function ExtractFileName(ByVal strFN As String) As String
```

```
Dim varI1 As Variant
Dim varI2 As Variant
```

```
' find the last "\" in the string
varI1 = 0
```

```
Do
```

```

    varI2 = varI1
    varI1 = InStr(varI2 + 1, strFN, "\")
    Loop Until varI1 = 0

    ExtractFileName = Mid(strFN, varI2 + 1, Len(strFN) - varI2)

```

5 End Function

' extracts the file name sans extension from a path/filename string
Function ExtractFileNameNoExt(ByVal strFN As String) As String

```

    strFN = ExtractFileName(strFN)

```

10 Dim varI1 As Variant
 Dim varI2 As Variant

```

    ' find the last "." in the string
    varI1 = 0

```

```

    Do
        varI2 = varI1
        varI1 = InStr(varI2 + 1, strFN, ".")
    Loop Until varI1 = 0

```

```

    ExtractFileNameNoExt = Mid(strFN, 1, varI2 - 1)

```

End Function

' extracts the family name - everything up to \$R
20 Function ExtractFamilyName(ByVal strFN As String) As String

```

    strFN = ExtractFileName(strFN)

```

```

    Dim varI As Variant

```

```

    ' find "$R" in the string
    varI = InStr(1, strFN, "$R")

```

25 If varI > 0 Then
 ExtractFamilyName = Mid(strFN, 1, varI - 1)
 End If

End Function

30 ' extracts the key, meaning \$R and everthing up to the .
 Function ExtractFamilyKey(ByVal strFN As String) As String

```
strFN = ExtractFileName(strFN)
```

```
Dim varI As Variant  
Dim varI1 As Variant  
Dim varI2 As Variant
```

```
5 ' find "$R" in the string  
varI = InStr(1, strFN, "$R")
```

```
' find the last "." in the string  
varI1 = 0
```

```
Do  
10 varI2 = varI1  
varI1 = InStr(varI2 + 1, strFN, ".")  
Loop Until varI1 = 0
```

```
ExtractFamilyKey = Mid(strFN, varI, varI2 - varI)
```

```
End Function
```

```
15 ' trim nulls off the end of a string  
Function TrimAtFirstNull(ByVal strS As String) As String
```

```
Dim varI As Variant
```

```
varI = InStr(1, strS, Chr(0))  
TrimAtFirstNull = left(strS, varI - 1)
```

```
20 End Function
```

```
' returns a string with all instances of strFrom replaced  
' with strTo in string strS
```

```
Function ReplaceAll(ByVal strS As String, ByVal strFrom As String, _  
ByVal strTo As String) As String
```

```
25 Dim varI As Variant  
Dim intL As Integer
```

```
Do
```

```
varI = InStr(1, strS, strFrom)
```

```
If varI > 0 Then ' found strFrom
```

```
30 intL = Len(strS)  
strS = left(strS, varI - 1) & strTo & _  
right(strS, intL - Len(strFrom) - varI + 1)  
End If
```

Loop Until varI = 0

ReplaceAll = strS

End Function

' returns the name of indexed string variables

5 Function GetIndexedName(ByVal strName As String, _
ByVal intI As Integer) As String

GetIndexedName = strName & "." & Trim(Str(intI))

End Function

10 ' Prolog shuts down when this file is created
Sub CreateKillFile()

Open HALT_FN For Output As #10
Print #10, "Halt!"
Close #10

15 End Sub

' Delete the kill file

Sub DestroyKillFile()

On Error Resume Next ' if it's not there, Kill will produce an error
Kill HALT_FN
err.Clear

20 End Sub


```
' MTAPI.BAS
Attribute VB_Name = "MTAPI"
'mtapi.bas 4.0
```

```
5  '(c) Copyright 1992-1999 by Design Science, Inc. All rights reserved
' with the exception that registered MathType owners may alter these
' macros for use by themselves and other registered MathType owners
' provided that:
' 1) The alterations are summarized in a comment directly below this
10 ' copyright notice. The comment should start with the words
' "Modified by" and include the name of the person altering the
' macros, the date of alteration, and that person's email address
' (if available).
' 2) Persons altering the macros notify Design Science of the nature
15 ' of any changes they have made.
' These provisions may help us help other customers, and will help us
' continue to provide quality products for you in the future.
```

```
' version # of this API
20 Public Const MTAPI_VERSION = 4

' maximum length of file paths, names, etc.
Public Const MTAPI_MAX_PATH = 260
```

```
' Picture specifier
Public Type MTAPI_PICT
25   mm    As Long
   xExt  As Long
   yExt  As Long
   hMF   As Long
End Type
```

```
30 Public Type RECT
   left  As Long
   top   As Long
   right As Long
   bottom As Long
35 End Type
```

```
' Picture dimensions
Public Type MTAPI_DIMS
   baseline As Integer ' dist of baseline from bottom (points)
   bounds   As RECT    ' bounding rectangle (points)
```

End Type

' return codes from MT DLL API

' success, no error

Public Const mtOK = 0

5 ' equation OLE 1.0 object on clipboard

Public Const mtOLE_EQUATION = 1

' Windows metafile equation graphic (not OLE object) on clipboard

Public Const mtWMF_EQUATION = 2

' Macintosh PICT equation graphic (not OLE object) on clipboard

10 Public Const mtMAC_PICT_EQUATION = 4

' equation OLE 2.0 object on clipboard

Public Const mtOLE2_EQUATION = 8

' error return codes

' can't find MathType application

15 Public Const mtMT_NOT_FOUND = -1

' can't run the MathType application

Public Const mtMT_CANT_RUN = -2

' the MathType application is the wrong version

Public Const mtMT_BAD_VERSION = -3

20 ' the MathType application is already in use

Public Const mtMT_IN_USE = -4

' the MathType application is not running (i.e. unexpectedly aborted)

Public Const mtMT_NOT_RUNNING = -5

' time ran out waiting for the MathType application to start up

25 Public Const mtRUN_TIMEOUT = -6

' not equation on clipboard

Public Const mtNOT_EQUATION = -7

' file does not exist or bad pathname

Public Const mtFILE_NOT_FOUND = -8

30 ' insufficient memory

Public Const mtMEMORY = -9

' bad file

Public Const mtBAD_FILE = -10

' requested data does not exist

35 Public Const mtDATA_NOT_FOUND = -11

' too many server session open

Public Const mtTOO_MANY_SESSIONS = -12

' could not perform one or more subs

Public Const mtSUBSTITUTION_ERROR = -13

40 ' could not perform translation

Public Const mtTRANSLATOR_ERROR = -14

```
' could not set preferences, or invalid preference string
Public Const mtPREFERENCE_ERROR = -15
' other error
Public Const mtERROR = -9999
```

```
5 ' options values for MTInitAPI
Public Const mtinitLAUNCH_AS_NEEDED = 0
Public Const mtinitLAUNCH_NOW = 1
```

```
' options values for MTGetTranslatorsInfo
Public Const mttrnCOUNT = 1
10 Public Const mttrnMAX_NAME = 2
Public Const mttrnMAX_DESC = 3
Public Const mttrnMAX_FILE = 4
Public Const mttrnOPTIONS = 5
```

```
' options values for MTXFormAddVarSub
15 Public Const mtxfmSUBST_ALL = 0
Public Const mtxfmSUBST_ONE = 1
```

```
' find/replace types for MTXFormAddVarSub substitutions
Public Const mtxfmVAR_SUB_BAD = -1
Public Const mtxfmVAR_SUB_PLAIN_TEXT = 0
20 Public Const mtxfmVAR_SUB_MTEF_TEXT = 1
Public Const mtxfmVAR_SUB_MTEF_BINARY = 2
Public Const mtxfmVAR_SUB_DELETE = 3
```

```
' replace style for MTXFormAddVarSub substitutions when replaceType =
mtxfmVAR_SUB_PLAIN_TEXT
25 Public Const mtxfmSTYLE_TEXT = 1
Public Const mtxfmSTYLE_FUNCTION = 2
Public Const mtxfmSTYLE_VARIABLE = 3
Public Const mtxfmSTYLE_LCGREEK = 4
Public Const mtxfmSTYLE_UCGREEK = 5
30 Public Const mtxfmSTYLE_SYMBOL = 6
Public Const mtxfmSTYLE_VECTOR = 7
Public Const mtxfmSTYLE_NUMBER = 8
```

```
' options values for MTXFormSetPrefs
Public Const mtxfmPREF_EXISTING = 1
35 Public Const mtxfmPREF_MTDEFAULT = 2
Public Const mtxfmPREF_USER = 3
Public Const mtxfmPREF_LAST = 3
```

```
' options values for MTXFormSetTranslator
```

```

Public Const mtxfmTRANSL_INC_NONE = 0
Public Const mtxfmTRANSL_INC_NAME = 1
Public Const mtxfmTRANSL_INC_DATA = 2
Public Const mtxfmTRANSL_INC_MTDEFAULT = 4

5   ' return values from MTXFormGetStatus
Public Const mtxfmSTAT_PREF = -3
Public Const mtxfmSTAT_TRANSL = -2
Public Const mtxfmSTAT_ACTUAL_LEN = -1

' data sources/destinations for MTXFormEqn
10  Public Const mtxfmPREVIOUS = -1
Public Const mtxfmCLIPBOARD = -2
Public Const mtxfmLOCAL = -3

' data formats for MTXFormEqn
Public Const mtxfmMTEF = 4
15  Public Const mtxfmHMTEF = 5
Public Const mtxfmPICT = 6
Public Const mtxfmTEXT = 7
Public Const mtxfmHTEXT = 8

' option values for MTSetMTPrefs
20  Public Const mtpfMODE_NEXT_EQN = 1
Public Const mtpfMODE_MTDEFAULT = 2
Public Const mtpfMODE_INLINE = 4

' MT API functions
Public Declare Function MTAPIVersion Lib "mt4" (ByVal api As Integer) As Long
25  Public Declare Function MTInitAPI Lib "mt4" (ByVal options As Integer, ByVal timeout As Integer) As Long
Public Declare Function MTTermAPI Lib "mt4" () As Long
Public Declare Function MTClearClipboard Lib "mt4" () As Long
Public Declare Function MTEquationOnClipboard Lib "mt4" () As Long
30  Public Declare Function MTXFormReset Lib "mt4" () As Long
Public Declare Function MTXFormAddVarSub Lib "mt4" ( _
    ByVal options As Integer, _
    ByVal findType As Integer, ByVal find As String, ByVal findLen As Long, _
    ByVal replaceType As Integer, ByVal replace As String, ByVal replaceLen As Long, _
35  ByVal replaceStyle As Integer _
) As Long
Public Declare Function MTXFormSetTranslator Lib "mt4" (ByVal options As Integer, _
    ByVal transName As String) As Long
Public Declare Function MTXFormSetPrefs Lib "mt4" (ByVal prefType As Integer, ByVal prefStr As String) As Long
40

```

```

Public Declare Function MTSetMTPrefs Lib "mt4" (ByVal mode As Integer, ByVal prefs As
String, _
    ByVal timeout As Integer) As Long
Public Declare Function MTXFormEqn Lib "mt4" ( _
5   ByVal src As Integer, ByVal srcFmt As Integer, ByVal srcData As String, ByVal srcDataLen
As Long, _
    ByVal dst As Integer, ByVal dstFmt As Integer, ByVal dstData As String, ByVal dstDataLen
As Long, _
    ByRef dims As MTAPI_DIMS) As Long
10 Public Declare Function MTXFormGetStatus Lib "mt4" (ByVal index As Integer) As Long

```

→

VBSCA -16-

```

'MTDeclaration.bas
Attribute VB_Name = "MTDeclarations"
'=====
'Windows API declarations
'=====
Public Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal hwnd As Long,
ByVal lpHelpFile As String, ByVal wCommand As Long, ByVal dwData As Long) As Long
Public Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal
lpLibFileName As String) As Long
Public Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long) As Long
Public Declare Function LoadString Lib "user32" Alias "LoadStringA" (ByVal hInstance As
Long, ByVal wID As Long, ByVal lpBuffer As String, ByVal nBufferMax As Long) As Long
Public Declare Function GetLocaleInfo Lib "kernel32" Alias "GetLocaleInfoA" (ByVal Locale
As Long, ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As
Long
Public Declare Function GetEnvironmentVariable Lib "kernel32" Alias
"GetEnvironmentVariableA" (ByVal lpName As String, ByVal lpBuffer As String, ByVal nSize
As Long) As Long
Public Declare Function SetEnvironmentVariable Lib "kernel32" Alias
"SetEnvironmentVariableA" (ByVal lpName As String, ByVal lpValue As String) As Long
Public Declare Function GetTickCount Lib "kernel32" () As Long

'=====
' Constants for use in Windows API calls
'=====
'----- Used by GetLocaleInfo -----
' values for LCType (locale info requested) - used in MTLib.InitLocaleStrs
Public Const Locale_SLanguage As Long = &H2
Public Const Locale_SEngLanguage As Long = &H1001

'=====
' Constants for use in Help calls
'=====
Public Const hlpMSWDPreferences_Dialog = 117
Public Const hlpMSWDEquation_Number_Format_Dialog = 6300
Public Const hlpMSWDFormat_Equations_Dialog = 6500
Public Const hlpMSWDInsert_Equation_Section_Dialog = 114
Public Const hlpMSWDFormat_Equation_Section_Dialog = 116
Public Const hlpMSWDSets_Equation_Preferences_Dialog = 37
Public Const hlpMSWDConvert_Equations_Dialog = 44
Public Const hlpMSWDInsert_Equation_Number_Dialog = 118
Public Const hlpMSWDInsert_Requation_Ref_Dialog = 119

Public Const hlpMSWDWT_SetEqnPrefs = 122

```

```

Public Const hlpMSWDWT_InlineEqn = 123
Public Const hlpMSWDWT_CenteredEqn = 124
Public Const hlpMSWDWT_CenteredNumberedEqn = 125
Public Const hlpMSWDWT_EqnNumber = 126
5 Public Const hlpMSWDWT_EqnRef = 127
Public Const hlpMSWDWT_EqnSec = 128
Public Const hlpMSWDWT_ModEqnSec = 129
Public Const hlpMSWDWT_FormatEqnNum = 130
Public Const hlpMSWDWT_ConvertEqn = 131
10 Public Const hlpMSWDWT_FormatEqn = 132
Public Const hlpMSWDWT_UpdateEqn = 133
'=====
' Constants for use in the MathType Commands
'=====
15 '----- Numbers we compare against with MTAPIvers -----
Public Const mtversMajVerHi = 1279 '0x04ff
Public Const mtversMajVerLo = 1024 '0x0400
Public Const mtversMinVer = 1024 '0x0400

'----- Registry location codes -----
20 Public Const mtreg_MT_LANG_LOCATION As String =
"HKEY_CURRENT_USER\Software\Design Science\DSMT4\Config" 'Registry entry for
MathType's curent language
Public Const mtreg_MT_LANG_KEY As String = "AppLang" 'registry key for MathType's
curent language
25 Public Const mtreg_MT_PROGDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories" 'Registry
entry for MathType's directory
Public Const mtreg_MT_PROGDIR_KEY As String = "ProgDir" 'registry key for MathType's
directory
30 Public Const mtreg_MT_LANGUAGEDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories" 'Registry
entry for MathType's language support files directory
Public Const mtreg_MT_LANGUAGEDIR_KEY As String = "LastLangDir" 'registry key for
MathType's language support files directory
35 Public Const mtreg_MT_HELPDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories" 'Registry
entry for MathType's help file directory
Public Const mtreg_MT_HELPDIR_KEY As String = "LastHelpDir" 'registry key for
MathType's help file directory
40 Public Const mtreg_MT_HELPFILE_LOCATION As String =
"HKEY_CURRENT_USER\Software\Design Science\DSMT4\Config" 'Registry entry for
MathType's help file name
Public Const mtreg_MT_HELPFILE_KEY As String = "HelpFile" 'registry key for
MathType's help file name

```

```

Public Const mtreg_MT_SYSTEMDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories" 'Registry
entry for MathType's system directory
Public Const mtreg_MT_SYSTEMDIR_KEY As String = "LastAppSystemDir" 'registry key
5 for MathType's system directory
Public Const mtreg_MT_PREFDIR_LOCATION As String =
"HKEY_LOCAL_MACHINE\SOFTWARE\Design Science\DSMT4\Directories" 'Registry
entry for MathType's preferences folder
Public Const mtreg_MT_PREFDIR_KEY As String = "LastPrefsDir" 'registry key for
10 MathType's system directory
Public Const mtreg_MT_WORDCMDSDS_LOCATION As String =
"HKEY_CURRENT_USER\SOFTWARE\Design Science\DSMT4\WordCommands"
'Registry entry for MathType's Word Commands data

Public Const mtreg_MT_WORD_CONVFROM As String = "ConvertFrom" 'ConvertFrom
15 key
Public Const mtreg_MT_WORD_CONVTO As String = "ConvertTo" 'ConvertTo key
Public Const mtreg_MT_WORD_CONVMISC As String = "ConvertMisc" 'ConvertMisc key
Public Const mtreg_MT_WORD_CONVTRANS As String = "ConvertTranslator"
'ConvertTranslator key

20 Public Const mtreg_MT_WORD_DONTSHOW_EQNREFDLG As String =
"NoInsertEqnRefDlg" 'Don't Show Insert Eqn Ref dialog key
Public Const mtreg_MT_WORD_DONTSHOW_SLOWEQNUPDATE As String =
"NoSlowUpdateEqnDlg" 'Don't Show Insert Eqn Ref dialog key

Public Const mtreg_MT_WORD_DONTSHOW_LANGDLLERROR As String =
25 "NoLanguageDLLError" 'Don't show Missing Lang DLL error key

'----- Strings used in MT text equations (TeX and MathML) -----
Public Const mttexeqn_START As String = "% MathType!" 'The identifier at the beginning
of MathType translator text equations
Public Const mttexeqn_END As String = "% MathType!End!" 'The identifier at the end of
30 MathType translator text equations

'----- Property names -----
Public Const mtprop_USE_MATHTYPE_PREFS As String = "MTUseMTPrefs" 'The
name of the Document Property that indicates to use MathType's prefs for new equations
Public Const mtprop_PREFERENCES As String = "MTPreferences" 'Contains the
35 doc's settings for new equations
Public Const mtprop_PREFERENCES_FILE As String = "MTPreferenceSource" 'Contains
the doc's settings for new equations
Public Const mtprop_NUMBER_PREFS As String = "MTEquationNumber" 'Contains
the current equation number format preferences
40 Public Const mtprop_DEFER_FIELD_UPDATE As String = "MTDeferFieldUpdate"

```


'Controls field updating

Public Const mtpropEQUATION_SECTION_CHECKED As String = "MTEquationSection"

'Indicates if eqn section number is 0 check has been made

Public Const mtprop_EQNREFPANE As String = "MTEqnRefPane" 'Pane number containing
insertion point where ref. is to be placed

'----- AutoText entry names -----

Public Const mtautotext_MT3_EQN_NUMBER_FORMAT As String =

"ZMTEqnNumFormatPrefs" 'The name of old Autotext entry that held MathType3's equation
number format prototype

'----- MathType OLE data -----

Public Const mtole_PROGID As String = "Equation.DSMT4" 'OLE Prog ID used to identify
MathType 4

'----- Style names -----

Public Const mtstyle_EQUATION_SECTION As String = "MTEquationSection" 'Style used for
eqn. section names

Public Const mtstyle_DISPLAY_EQUATION As String = "MTDisplayEquation" 'Style used
for display equations

'----- Misc. constants -----

'Constants used to specify 'curent selection' or 'whole document'

Public Const mt_RANGE_DOCUMENT = 0

Public Const mt_RANGE_SELECTION = 1

'Constants used by MTMsgBox

Public Const mt_MBYESNO = 1

Public Const mt_MBYESNOCANCEL = 2

Public Const mt_MBYES = 1

Public Const mt_MBNO = 2

Public Const mt_MBCANCEL = 3

'Flag bit for MTLib.SaveWordState()

Public Const mt_SWS_TRACKCHANGES = 1

Public Const mt_SWS_SMART_CUTPASTE = 2

Public Const mt_SWS_TYPING_REPLACE_SELECTION = 4

```
' MTUtil.bas
Attribute VB_Name = "MTUtil"
'MTUtil: 4.0
```

```
'=====
5  '(c) Copyright 1992-1999 by Design Science, Inc. All rights reserved
' with the exception that registered MathType owners may alter these
' macros for use by themselves and other registered MathType owners
' provided that:
' 1) The alterations are summarized in a comment directly below this
10 ' copyright notice. The comment should start with the words
' "Modified by" and include the name of the person altering the
' macros, the date of alteration, and that person's email address
' (if available).
' 2) Persons altering the macros notify Design Science of the nature
15 ' of any changes they have made.
' These provisions may help us help other customers, and will help us
' continue to provide quality products for you in the future.
'=====
```

```
'This macro contains subroutines used by other Design Science macros
'=====
```

```
Option Explicit
```

```
'Public Sub Main()
' MsgBox MTUtil.GetUserString("!1600This contains a library of functions shared by
MathType's macros."), _
25 ' vbOKOnly, MTUtil.GetUserString("!1601MTUtil Macro")
'End Sub
```

```
'=====
' CheckMTDLLVersion()
'Checks the MT DLL version. If it's a bad version, we display an
30 'error and return 0. If we can still run, returns nonzero
'=====
```

```
Public Function CheckMTDLLVersion()
```

```
Dim errorflag
Dim dllver
35 Dim msg$
Dim myResult
```

```
errorflag = 0
CheckMTDLLVersion = 1 'assume success to start
```

```
'init the API
```

```

If MTInitAPI(mtinitLAUNCH_AS_NEEDED, 30) <> 0 Then
    msg$ = MTUtil.GetUserString("!1606The MathType commands could not communicate
with MathType. There was a problem starting the API. Please be sure that MathType is properly
installed.")

```

```

5      CheckMTDLLVersion = 0
      errorflag = 1

```

```

Else
    'get the API Version
    dllver = MTAPIVersion(MTAPI_VERSION)

```

```

10    'check the version against our constants

```

```

    If (dllver > mtversMajVerHi) Or (dllver < mtversMajVerLo) Then
        msg$ = MTUtil.GetUserString("!1607The version of this macro doesn't match the
version of MathType's DLL. Reinstall MathType to fix this condition.")

```

```

    CheckMTDLLVersion = 0
    errorflag = 1

```

```

15    ElseIf (dllver < mtversMinVer) Then
        msg$ = MTUtil.GetUserString("!1608A more recent version of MathType's DLL is
required to use this macro. Reinstall MathType to fix this condition.")

```

```

    CheckMTDLLVersion = 0
    errorflag = 1

```

```

    End If
End If

```

```

    If (errorflag = 1) Then 'report error condition
        MsgBox msg$, vbCritical, MTUtil.GetUserString("!1609MathType Commands for
Microsoft Word Error")

```

```

25    End If
End Function

```

```

'=====
'      GetUserString$
'=====

```

```

30 Public Function GetUserString$(EnglishString$)
    'simply return the English version (strip "!nnnn" from start)
    GetUserString$ = right(EnglishString$, Len(EnglishString$) - 5)
End Function

```

```

35 '=====
'      GetMathTypeDir$
'      Gets the location of MathType from the registry
'=====

```

```

40 Public Function GetMathTypeDir$()
    Dim path$

```

```

'get the location of Mathtype from the registry
path$ = System.PrivateProfileString("", mtreg_MT_PROGDIR_LOCATION,
mtreg_MT_PROGDIR_KEY)

```

```

'return the results
5   GetMathTypeDir$ = path$
End Function

```

```

=====
'
'           WritePermSetting
=====

```

```

10  'Writes key/value pair to permanent location, ie Windows registry.
'Used when data needs to be saved whose scope is larger than a document.
Public Sub WritePermSetting(key$, data$)
    System.PrivateProfileString("", mtreg_MT_WORDCMDS_LOCATION, key$) = data$
End Sub

```

```

15  =====
'
'           ReadPermSetting$
=====

```

```

'Reads key's value from the permanent location, ie Windows registry.
'Used when data needs to be saved whose scope is larger than a document.
20  Public Function ReadPermSetting$(key$)
    ReadPermSetting$ = System.PrivateProfileString("", mtreg_MT_WORDCMDS_LOCATION,
key$)
End Function

```

```

25  =====
'
'           SetNextTXFormPrefs
'Sets prefs that MathType will use for the next transformed equation.
'Returns MTXFormSetPrefs result code.
=====

```

```

Function SetNextTXFormPrefs(prefStr$)

```

```

30   Dim stat

```

```

'set preferences for next transformed equation
stat = MTXFormSetPrefs(mtxfmPREF_USER, prefStr$)

```

```

If stat <> 0 Then
    MsgBox MTUtil.GetUserString("!1100There was a problem sending your equation
35  preferences for " _
    + "this document to MathType. This equation will use MathType's " _
    + "'New Equation' preferences."), vbExclamation, _
    MTUtil.GetUserString("!1101MathType Preferences Problem")
End If

```

```

    SetNextTXFormPrefs = stat
End Function

```

```

'=====
'          SetPrefsForNextEqn
5 'Sets prefs that MathType will use for the next new equation.
'Returns MTSetMTPrefs result code.
'=====

```

```

Public Function SetPrefsForNextEqn(prefStr$, inline As Boolean)

```

```

    Dim stat
10    Dim options As Integer

```

```

    options = mtpfMODE_NEXT_EQN
    If inline Then options = options + mtpfMODE_INLINE
    'set preferences for next transformed equation
    stat = MTSetMTPrefs(options, prefStr$, -1)

```

```

15    If stat <> 0 Then
        MsgBox MTUtil.GetUserString("!1100There was a problem sending your equation
preferences for " _
            + "this document to MathType. This equation will use MathType's " _
            + "'New Equation' preferences."), vbExclamation, _
20        MTUtil.GetUserString("!1101MathType Preferences Problem")
    End If

```

```

    SetPrefsForNextEqn = stat
End Function

```

```

'=====
'          IsEquationProgID
25 'Returns 1 if the progID is a MathType/EE OLE1 progID.
'Returns 2 if the progID is a MathType/EE OLE2 progID.
'Returns 0 if not a recognized progID.
'=====

```

```

30 Public Function IsEquationProgID(progID$) As Long

```

```

    Dim uProgID$
    uProgID$ = UCase(progID$)

```

```

    If uProgID$ = "EQUATION" Then
        IsEquationProgID = 1
35    ElseIf InStr(1, uProgID$, "EQUATION.", vbBinaryCompare) = 1 Then
        IsEquationProgID = 2
    Else
        IsEquationProgID = 0
    End If

```

```

40 End Function

```

```

'      TransformGraphicEquation
'Attempts to transform the graphic on the clipboard into an equation.
'Resulting format depends on how MathType has been configured by a
'previous call to MTXFormSetTranslator.
5 'The transformed equation is left on the clipboard.
'If OK, returns mtOK
'If not an equation, or an error occurred, returns mtNOT_EQUATION
'=====

Public Function TransformGraphicEquation() As Long
10   TransformGraphicEquation = mtNOT_EQUATION

   'Use API call to check clipboard contents first
   If MTEquationOnClipboard() = mtNOT_EQUATION Then
       Exit Function
   End If

15   TransformGraphicEquation = TransformEquation()
End Function
'=====

'      TransformEquation
'Attempts to transform the item on the clipboard into an equation.
20 'Resulting format depends on how MathType has been configured by a
'previous call to MTXFormSetTranslator.
'The transformed equation is left on the clipboard.
'If OK, returns mtOK
'If not an equation, or an error occurred, returns mtNOT_EQUATION
25 '=====

Public Function TransformEquation() As Long
   Dim stat As Long
   Dim dummyStr1$, dummyStr2$
   Dim dummyDims As MTAPI_DIMS

30   On Error GoTo err

   stat = mtNOT_EQUATION

   'as long as everything's OK, update the equation
   'set aside some buffers
   dummyStr1$ = Space(1)
35   dummyStr2$ = Space(1)
   With dummyDims
       .baseline = 0
       .bounds.bottom = 0
       .bounds.left = 0
40       .bounds.right = 0

```

```
        .bounds.top = 0
```

```
End With
```

```
'do the update
```

```
stat = MTXFormEqn(mtxfmCLIPBOARD, mtxfmTEXT, dummyStr1$, 1, _
```

```
    mtxfmCLIPBOARD, mtxfmTEXT, dummyStr2$, 1, dummyDims)
```

```
If stat < 0 Then
```

```
    stat = mtNOT_EQUATION
```

```
End If
```

```
GoTo Bye
```

```
10 err:
```

```
    If err.Number = 5690 Or err.Number = 4198 Then
```

```
        'the user has revisions on, and this is an old revision that has been deleted
```

```
        stat = -2
```

```
        Resume Bye
```

```
15    Else
```

```
        err.Raise err.Number
```

```
        Stop
```

```
    End If
```

```
Bye:
```

```
20    TransformEquation = stat
```

```
End Function
```

DeleteDocProperty

```
25 'deletes document property, OK to call if it doesn't exist
```

```
Public Function DeleteDocProperty(doc As Document, prop$)
```

```
    On Error GoTo Error
```

```
    doc.CustomDocumentProperties(prop$).Delete
```

```
Error:
```

```
30 End Function
```

DocPropertyExists

```
'returns True if the active document contains the custom doc property
```

```
35 Public Function DocPropertyExists(propName$) As Boolean
```

```
    Dim name$
```

```
    DocPropertyExists = False
```

```
    On Error GoTo Error
```

```
    name$ = ActiveDocument.CustomDocumentProperties(propName$).name
```

```
DocPropertyExists = True
Error:
End Function
```

```
5  '=====
   '          Delay
   'Pauses execution for timeout (in milliSecs)
   '=====

Public Sub Delay(timeout As Long)
    Dim start As Long
10    start = GetTickCount()
    Do While (GetTickCount() < (start + timeout))
        DoEvents ' Yield to other processes.
    Loop
End Sub
15 →
```



```
' Timer.bas
Attribute VB_Name = "Module1"
Option Explicit
```

```
5 Declare Function SetTimer Lib "user32" (ByVal hWnd As Long, _
    ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerProc As Long) _
    As Long
```

```
Declare Function KillTimer Lib "user32" (ByVal hWnd As Long, _
    ByVal nIDEvent As Long) As Long
```

```
10 Public gProlog As Prolog
    Public gTimerID As Long
```

```
' called by SolveConstraintsRandomly in Prolog.cls
Public Sub SolveAsync()
```

```
15     ' calls TimerCallback when timer runs out (it's set for 0, so it
    ' runs out immediately. TimerCallback, and anything called by
    ' TimerCallback, run async.
    gTimerID = SetTimer(0, 0, 1000, AddressOf TimerCallback)
```

```
End Sub
```

```
20 Public Sub TimerCallback(ByVal hWnd As Long, ByVal uMsg As Long, ByVal idEvent As
    Long, ByVal dwTime As Long)
```

```
    KillTimer 0, gTimerID
```

```
    gProlog.SolveConstraintsAsync ' in Prolog.cls
```

```
25 End Sub
```

```

' Constraint.frm
VERSION 5.00
Object = "{BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0"; "TABCTL32.OCX"
Begin VB.Form frmConstraints
5   BorderStyle   = 4 'Fixed ToolWindow
    Caption      = "Create or Change Constraints"
    ClientHeight = 6405
    ClientLeft   = 45
    ClientTop    = 285
10   ClientWidth  = 6285
    LinkTopic    = "Form1"
    MaxButton    = 0 'False
    MinButton    = 0 'False
    ScaleHeight  = 6405
15   ScaleWidth   = 6285
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
Begin TabDlg.SSTab sstConstraintTool
    Height       = 3375
20   Left        = 240
    TabIndex     = 5
    Top         = 1080
    Width       = 4455
    _ExtentX    = 7858
25   _ExtentY    = 5953
    _Version    = 393216
    TabHeight   = 520
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
    Name        = "MS Sans Serif"
30   Size       = 8.25
    Charset    = 0
    Weight     = 400
    Underline  = 0 'False
    Italic     = 0 'False
35   Strikethrough = 0 'False
EndProperty
    TabCaption(0) = "Operators"
    TabPicture(0) = "Constraint.frx":0000
    Tab(0).ControlEnabled= -1 'True
40   Tab(0).Control(0)= "cmdElseIf"
    Tab(0).Control(0).Enabled= 0 'False
    Tab(0).Control(1)= "cmdElse"
    Tab(0).Control(1).Enabled= 0 'False
    Tab(0).Control(2)= "cmdThen"

```

```

Tab(0).Control(2).Enabled= 0 'False
Tab(0).Control(3)= "cmdIf"
Tab(0).Control(3).Enabled= 0 'False
Tab(0).Control(4)= "cmdLessThanOrEqualTo"
5 Tab(0).Control(4).Enabled= 0 'False
Tab(0).Control(5)= "cmdGreaterThanEqualTo"
Tab(0).Control(5).Enabled= 0 'False
Tab(0).Control(6)= "cmdLessThan"
Tab(0).Control(6).Enabled= 0 'False
10 Tab(0).Control(7)= "cmdGreaterThan"
Tab(0).Control(7).Enabled= 0 'False
Tab(0).Control(8)= "cmdNotEqual"
Tab(0).Control(8).Enabled= 0 'False
Tab(0).Control(9)= "cmdAbs"
15 Tab(0).Control(9).Enabled= 0 'False
Tab(0).Control(10)= "cmdFactorial"
Tab(0).Control(10).Enabled= 0 'False
Tab(0).Control(11)= "cmdExponent"
Tab(0).Control(11).Enabled= 0 'False
20 Tab(0).Control(12)= "cmdQuotient"
Tab(0).Control(12).Enabled= 0 'False
Tab(0).Control(13)= "cmdList"
Tab(0).Control(13).Enabled= 0 'False
Tab(0).Control(14)= "cmdModulus"
25 Tab(0).Control(14).Enabled= 0 'False
Tab(0).Control(15)= "cmdEqual"
Tab(0).Control(15).Enabled= 0 'False
Tab(0).Control(16)= "cmdDivide"
Tab(0).Control(16).Enabled= 0 'False
30 Tab(0).Control(17)= "cmdMultiply"
Tab(0).Control(17).Enabled= 0 'False
Tab(0).Control(18)= "cmdMinus"
Tab(0).Control(18).Enabled= 0 'False
Tab(0).Control(19)= "cmdPlus"
35 Tab(0).Control(19).Enabled= 0 'False
Tab(0).Control(20)= "cmdParens"
Tab(0).Control(20).Enabled= 0 'False
Tab(0).ControlCount= 21
TabCaption(1) = "Variables"
40 TabPicture(1) = "Constraint.frx":001C
Tab(1).ControlEnabled= 0 'False
Tab(1).Control(0)= "cboVariableNames"
Tab(1).Control(0).Enabled= 0 'False
Tab(1).Control(1)= "cmdInsertVN"
45 Tab(1).Control(1).Enabled= 0 'False

```

```

Tab(1).ControlCount= 2
TabCaption(2) = "Functions"
TabPicture(2) = "Constraint.frx":0038
Tab(2).ControlEnabled= 0 'False
Tab(2).Control(0)= "cboFunction"
Tab(2).Control(0).Enabled= 0 'False
Tab(2).Control(1)= "cmdInsertFunction"
Tab(2).Control(1).Enabled= 0 'False
Tab(2).Control(2)= "txtFunctionDescription"
Tab(2).Control(2).Enabled= 0 'False
Tab(2).ControlCount= 3
Begin VB.CommandButton cmdParens
    Caption      = "("
    BeginProperty Font
        Name      = "MS Sans Serif"
        Size      = 9.75
        Charset   = 0
        Weight    = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left       = 2280
    TabIndex   = 32
    ToolTipText = "List"
    Top        = 1320
    Width      = 495
End
Begin VB.ComboBox cboFunction
    Height      = 315
    ItemData    = "Constraint.frx":0054
    Left       = -74400
    List        = "Constraint.frx":007C
    Style       = 2 'Dropdown List
    TabIndex   = 31
    ToolTipText = "Select a Prolog function from the list."
    Top        = 840
    Width      = 2175
End
Begin VB.CommandButton cmdInsertFunction
    Caption     = "Insert"
    Height      = 315
    Left       = -72120
    TabIndex   = 30

```

```
    ToolTipText = "Click here to insert this function into the constraint above at the current  
cursor position."
```

```
    Top        = 840  
    Width      = 855
```

```
End
```

```
Begin VB.TextBox txtFunctionDescription
```

```
    Height      = 1455  
    Left        = -74400  
    Locked      = -1 'True  
    MultiLine   = -1 'True  
    ScrollBars  = 2 'Vertical  
    TabIndex    = 29
```

```
    ToolTipText = "The description of the function appears in this window."  
    Top         = 1320  
    Width       = 3135
```

```
End
```

```
Begin VB.ComboBox cboVariableNames
```

```
    Height      = 315  
    ItemData    = "Constraint.frx":00EF  
    Left        = -74400  
    List        = "Constraint.frx":0117  
    Style       = 2 'Dropdown List  
    TabIndex    = 28
```

```
    ToolTipText = "Select a Prolog function from the list."  
    Top         = 1320  
    Width       = 2175
```

```
End
```

```
Begin VB.CommandButton cmdInsertVN
```

```
    Caption     = "Insert"  
    Height      = 315  
    Left        = -72120  
    TabIndex    = 27
```

```
    ToolTipText = "Click here to insert this variable name into the constraint above at the  
current cursor position."
```

```
    Top        = 1320  
    Width      = 855
```

```
End
```

```
Begin VB.CommandButton cmdPlus
```

```
    Caption     = "+"
```

```
BeginProperty Font
```

```
    Name       = "MS Sans Serif"  
    Size       = 9.75  
    Charset    = 0  
    Weight     = 400  
    Underline  = 0 'False
```

```

        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 480
    TabIndex     = 25
    ToolTipText  = "Plus"
    Top         = 840
    Width        = 495
End
Begin VB.CommandButton cmdMinus
    Caption      = "-"
    BeginProperty Font
        Name      = "MS Sans Serif"
        Size      = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 1080
    TabIndex     = 24
    ToolTipText  = "Minus"
    Top         = 840
    Width        = 495
End
Begin VB.CommandButton cmdMultiply
    Caption      = "*"
    BeginProperty Font
        Name      = "MS Sans Serif"
        Size      = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 1680
    TabIndex     = 23
    ToolTipText  = "Multiply"
    Top         = 840
    Width        = 495

```

End
Begin VB.CommandButton cmdDivide

Caption = "/"

BeginProperty Font

Name = "MS Sans Serif"

Size = 9.75

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

Height = 375

Left = 2280

TabIndex = 22

ToolTipText = "Divide"

Top = 840

Width = 495

End

Begin VB.CommandButton cmdEqual

Caption = "="

BeginProperty Font

Name = "MS Sans Serif"

Size = 9.75

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

Height = 375

Left = 480

TabIndex = 21

ToolTipText = "Equals"

Top = 1800

Width = 495

End

Begin VB.CommandButton cmdModulus

Caption = "%"

BeginProperty Font

Name = "MS Sans Serif"

Size = 9.75

Charset = 0

Weight = 400

Underline = 0 'False

```

        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 2880
    TabIndex     = 20
    ToolTipText  = "Modulo"
    Top         = 840
    Width        = 495
End
Begin VB.CommandButton cmdList
    Caption      = "([1,2])"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 2880
    TabIndex     = 19
    ToolTipText  = "List"
    Top         = 1320
    Width        = 1095
End
Begin VB.CommandButton cmdQuotient
    Caption      = "\"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 480
    TabIndex     = 18
    ToolTipText  = "Quotient"
    Top         = 1320
    Width        = 495

```



```

End
Begin VB.CommandButton cmdExponent
    Caption      = "^"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 375
    Left        = 3480
    TabIndex    = 17
    ToolTipText  = "Exponent"
    Top         = 840
    Width       = 495
End
Begin VB.CommandButton cmdFactorial
    Caption      = "!"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 375
    Left        = 1080
    TabIndex    = 16
    ToolTipText  = "Factorial"
    Top         = 1320
    Width       = 495
End
Begin VB.CommandButton cmdAbs
    Caption      = "||"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False

```

```

        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 1680
    TabIndex     = 15
    ToolTipText  = "Absolute value"
    Top          = 1320
    Width        = 495
End
10 Begin VB.CommandButton cmdNotEqual
    Caption      = "=/="
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size        = 9.75
        Charset     = 0
        Weight      = 400
        Underline   = 0 'False
        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 1080
    TabIndex     = 14
    ToolTipText  = "Does not equal"
    Top          = 1800
    Width        = 495
End
15 Begin VB.CommandButton cmdGreaterThan
    Caption      = ">"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size        = 9.75
        Charset     = 0
        Weight      = 400
        Underline   = 0 'False
        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Left        = 1680
    TabIndex     = 13
    ToolTipText  = "Greater than"
    Top          = 1800
    Width        = 495

```

```

End
Begin VB.CommandButton cmdLessThan
    Caption      = "<"
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 375
    Left        = 2280
    TabIndex    = 12
    ToolTipText  = "Less than"
    Top         = 1800
    Width       = 495
End
Begin VB.CommandButton cmdGreaterThanOrEqualTo
    Caption      = ">="
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 375
    Left        = 2880
    TabIndex    = 11
    ToolTipText  = "Greater than or equal to"
    Top         = 1800
    Width       = 495
End
Begin VB.CommandButton cmdLessThanOrEqualTo
    Caption      = "<="
    BeginProperty Font
        Name       = "MS Sans Serif"
        Size       = 9.75
        Charset    = 0
        Weight     = 400
        Underline  = 0 'False

```

```

        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
5    Left       = 3480
    TabIndex    = 10
    ToolTipText = "Less than or equal to"
    Top        = 1800
    Width      = 495
10   End
    Begin VB.CommandButton cmdIf
        Caption      = "if"
        BeginProperty Font
            Name       = "MS Sans Serif"
15         Size      = 9.75
            Charset    = 0
            Weight     = 400
            Underline  = 0 'False
            Italic     = 0 'False
            Strikethrough = 0 'False
20         EndProperty
        Height      = 375
        Left       = 480
        TabIndex    = 9
25         ToolTipText = "If"
        Top        = 2280
        Width      = 735
    End
    Begin VB.CommandButton cmdThen
30         Caption      = "then"
        BeginProperty Font
            Name       = "MS Sans Serif"
            Size      = 9.75
            Charset    = 0
35         Weight     = 400
            Underline  = 0 'False
            Italic     = 0 'False
            Strikethrough = 0 'False
        EndProperty
40         Height      = 375
        Left       = 1320
        TabIndex    = 8
        ToolTipText = "then"
        Top        = 2280
45         Width      = 735
    End

```

End
Begin VB.CommandButton cmdElse

Caption = "else"

BeginProperty Font

Name = "MS Sans Serif"

Size = 9.75

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

Height = 375

Left = 2160

TabIndex = 7

ToolTipText = "else"

Top = 2280

Width = 735

End

Begin VB.CommandButton cmdElseIf

Caption = "elseif"

BeginProperty Font

Name = "MS Sans Serif"

Size = 9.75

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

Height = 375

Left = 3000

TabIndex = 6

ToolTipText = "elseif"

Top = 2280

Width = 975

End

End

Begin VB.TextBox txtConstraint

Height = 315

Left = 240

TabIndex = 3

ToolTipText = "Enter the constraint here."

Top = 480

Width = 4455

```

End
Begin VB.TextBox txtComment
    Height      = 1335
    Left        = 240
    MultiLine   = -1 'True
    TabIndex    = 0
    Top         = 4800
    Width       = 4455
End
10 Begin VB.CommandButton cmdConOK
    Caption     = "OK"
    Default     = -1 'True
    Height      = 495
    Left        = 4920
    TabIndex    = 1
    ToolTipText = "Click here to save this constraint."
    Top         = 120
    Width       = 1215
End
20 Begin VB.CommandButton cmdConCancel
    Caption     = "Cancel"
    Height      = 495
    Left        = 4920
    TabIndex    = 2
    ToolTipText = "Click here to return without creating or modifying this constraint."
    Top         = 720
    Width       = 1215
End
Begin VB.Label lblComment
30 Caption     = "Comment"
    Height      = 255
    Left        = 240
    TabIndex    = 26
    Top         = 4560
    Width       = 1215
End
35 Begin VB.Label lblConstraints
    Caption     = "Constraint"
    Height      = 255
    Left        = 240
    TabIndex    = 4
    ToolTipText = "Click on the down arrow for function prototypes"
    Top         = 240
    Width       = 1695
45 End

```

```

End
Attribute VB_Name = "frmConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
5 Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

```

```
Private mbytAddEditFlag As Byte
```

```
Private mlstListBox As ListBox
```

```
10 Private mudtCon As Constraint
```

```
Private mudtModel As Model
```

```
Private mudtConType As ConstraintType
```

```
Private Enum ResourceStrings
```

```
    rcStartFunctions = 101
```

```
    rcEndFunctions = 125
```

```
    rcStartExplanations = 201
```

```
End Enum
```

```
Private mblnChangeFocus As Boolean
```

```
Public Property Let AddEditFlag(ByVal bytNewValue As Byte)
```

```
    mbytAddEditFlag = bytNewValue
```

```
End Property
```

```
Public Property Let ListBox(ByVal lstNewValue As ListBox)
```

```
    Set mlstListBox = lstNewValue
```

```
End Property
```

```
25 Public Property Let Constraint(ByVal udtNewValue As Constraint)
```

```
    Set mudtCon = udtNewValue
```

```
End Property
```

```
Public Property Let ConstraintType(ByVal udtNewValue As ConstraintType)
```

```
        mudtConType = udtNewValue
```

```
End Property
```

```
Public Property Let Model(ByVal udtNewValue As Model)
```

```
    Set mudtModel = udtNewValue
```

```
5 End Property
```

```
Private Sub cboFunction_Click()
```

```
    Dim intI As Integer
```

```
    For intI = 0 To cboFunction.ListCount - 1
```

```
10    If cboFunction = cboFunction.List(intI) Then
```

```
        txtFunctionDescription = LoadResString(intI + rcStartExplanations)
```

```
        Exit For
```

```
    End If
```

```
Next intI
```

```
15 If mbInChangeFocus Then
```

```
    txtConstraint.SetFocus
```

```
End If
```

```
20 End Sub
```

```
Private Sub cboVariableNames_Click()
```

```
    If mbInChangeFocus Then
```

```
        txtConstraint.SetFocus
```

```
    End If
```

```
25
```

```
End Sub
```

```
Private Sub cmdElse_Click()
```

```
    Call InsertText("else", 0)
```

```
30 End Sub
```

```
Private Sub cmdElseIf_Click()
```

```
    Call InsertText("elseif", 0)
```


End Sub

Private Sub cmdGreaterThan_Click()

Call InsertText(">", 0)

5 End Sub

Private Sub cmdGreaterThanEqualTo_Click()

Call InsertText(">=", 0)

End Sub

10 Private Sub cmdIf_Click()

Call InsertText("if", 0)

End Sub

Private Sub cmdParens_Click()

15 Call InsertText("()", 1)

End Sub

Private Sub cmdThen_Click()

Call InsertText("then", 0)

20 End Sub

Private Sub cmdInsertFunction_Click()

If cboFunction = "brandom()" Or cboFunction = "random()" Then

Call InsertText(cboFunction, 0)

25 Else

Call InsertText(cboFunction, 1)

End If

End Sub

30 Private Sub cmdInsertVN_Click()

Call InsertText(cboVariableNames, 0)

End Sub

Private Sub cmdLessThan_Click()

Call InsertText("<")

End Sub

Private Sub cmdLessThanOrEqualTo_Click()

Call InsertText("<=", 0)

End Sub

Private Sub cmdNotEqual_Click()

Call InsertText("≠", 0)

End Sub

Private Sub cmdPlus_Click()

Call InsertText("+")

End Sub

Private Sub cmdMinus_Click()

Call InsertText("-")

End Sub

Private Sub cmdMultiply_Click()

Call InsertText("*")

End Sub

Private Sub cmdDivide_Click()

Call InsertText("/")

End Sub

```
Private Sub cmdModulus_Click()
```

```
    Call InsertText("%")
```

```
End Sub
```

```
Private Sub cmdEqual_Click()
```

```
5    Call InsertText("=")
```

```
End Sub
```

```
Private Sub cmdList_Click()
```

```
    Call InsertText("[]", 2)
```

```
End Sub
```

```
10 Private Sub cmdQuotient_Click()
```

```
    Call InsertText("/")
```

```
End Sub
```

```
Private Sub cmdExponent_Click()
```

```
    Call InsertText("^")
```

```
15 End Sub
```

```
Private Sub cmdFactorial_Click()
```

```
    Call InsertText("!")
```

```
End Sub
```

```
Private Sub cmdAbs_Click()
```

```
20    Call InsertText("||", 1)
```

```
End Sub
```

```
Private Sub InsertText(ByVal strInsertedText As String, _  
    Optional ByVal intOffset As Integer = -1)
```

```
Dim strFront As String
Dim strBack As String
```

```
If intOffset = -1 Then intOffset = Len(strInsertedText) - 1
```

```
5 strFront = left(txtConstraint, txtConstraint.SelStart)
strBack = right(txtConstraint, Len(txtConstraint) - _
txtConstraint.SelStart - txtConstraint.SelLength)
```

```
txtConstraint = strFront & strInsertedText & strBack
10 txtConstraint.SetFocus
```

```
' move the cursor
txtConstraint.SelStart = Len(strFront) + Len(strInsertedText) - intOffset
```

```
15 End Sub
```

```
Private Sub Command3_Click()
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
' disable OK button if changes aren't allowed
20 If mudtModel.IsFrozen Then
cmdConOK.Enabled = False
Else
cmdConOK.Enabled = True
End If
```

```
25 Dim udtV As Variable
```

```
' load variable names into combo box
cboVariableNames.Clear
30 For Each udtV In mudtModel.Variables
Call cboVariableNames.AddItem(udtV.name)
Next udtV
```

```
If mbytAddEditFlag = aeEdit Then
35 txtConstraint = mudtCon.ConstraintString
txtComment = mudtCon.Comment
End If
```

```
'load functions into combo box
40 Dim intI As Integer
```

```

For intI = rcStartFunctions To rcEndFunctions
    cboFunction.List(intI - rcStartFunctions) = LoadResString(intI)
Next intI

```

```

5
    mblnChangeFocus = False
    If cboVariableNames.ListCount > 0 Then
        cboVariableNames.ListIndex = 0
    End If
10    cboFunction.ListIndex = 0
    mblnChangeFocus = True

```

```

End Sub

```

```

Private Sub cmdConOK_Click()

```

```

    If Len(txtConstraint) = 0 Then
15        Call MsgBox("Null constraints are not permitted", vbExclamation, "Error")
        Exit Sub
    End If

```

```

    If mbytAddEditFlag = aeEdit Then ' we're editing an old one
        ' update the constraint with new data from the form
20        Call mudtCon.Update(txtConstraint, mudtConType, txtComment)
        ' update the text in the list box
        mlstListBox.List(mlstListBox.ListIndex) = mudtCon.ConstraintString

```

```

    Else
        ' Add the new constraint
25        Set mudtCon = mudtModel.Constraints.Add(txtConstraint, True, _
            mudtConType, txtComment)
        With mlstListBox
            ' Add the new constraint to the list box
            Call .AddItem(mudtCon.ConstraintString)
30            ' Set ItemData to index value of the variable object
            .ItemData(.ListCount - 1) = mudtCon.index
            ' Check the check box
            .Selected(.ListCount - 1) = True

```

```

        End With
35    End If

    Call frmTCA.AddUndefinedVariables(txtConstraint)

```

```

    Unload Me

```

```

40
End Sub

```

Private Sub cmdConCancel_Click()

Unload Me

5 End Sub

VBSCA -49-

```

' EditConstraint.frm
VERSION 5.00
Begin VB.Form frmEditText
    BorderStyle   = 1 'Fixed Single
    ClientHeight  = 1455
    ClientLeft    = 45
    ClientTop     = 330
    ClientWidth   = 4785
    LinkTopic     = "Form1"
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 1455
    ScaleWidth    = 4785
    StartUpPosition = 3 'Windows Default
    Begin VB.CommandButton cmdEditTextOK
        Caption    = "OK"
        Default    = -1 'True
        Height     = 495
        Left       = 3360
        TabIndex   = 2
        Top        = 120
        Width      = 1215
    End
    Begin VB.CommandButton cmdEditTextnCancel
        Caption    = "Cancel"
        Height     = 495
        Left       = 3360
        TabIndex   = 1
        Top        = 720
        Width      = 1215
    End
    Begin VB.TextBox txtEditText
        Alignment   = 2 'Center
        Height      = 375
        Left        = 240
        TabIndex    = 0
        Top         = 120
        Width       = 2895
    End
End
Attribute VB_Name = "frmEditText"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True

```

5

10

	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2
--	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---


```

' Form1.frm
VERSION 5.00
Begin VB.Form Form1
    Caption       = "Form1"
    ClientHeight  = 4050
    ClientLeft    = 60
    ClientTop     = 345
    ClientWidth   = 5595
    LinkTopic     = "Form1"
    ScaleHeight   = 4050
    ScaleWidth    = 5595
    StartUpPosition = 3 'Windows Default
Begin VB.CommandButton Command1
    Caption       = "Clear"
    Height        = 1455
    Left          = 3720
    TabIndex      = 2
    Top           = 2520
    Width         = 1455
End
Begin VB.TextBox Text1
    Height        = 855
    Left          = 600
    TabIndex      = 1
    Text          = "Text1"
    Top           = 960
    Width         = 2175
End
Begin VB.CommandButton cmdRun
    Caption       = "Run"
    Height        = 1335
    Left          = 3720
    TabIndex      = 0
    Top           = 960
    Width         = 1455
End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

```

```
Private Sub cmdRun_Click()
```

```
    Dim udtP As New Prolog
```

```
    Dim lngR As Long
```

```
5    If udtP.StartProlog("hlp4lib.p4") = False Then  
        Call MsgBox("Prolog failure on startup", vbExclamation, "Error")  
    End If
```

```
10    Call udtP.AddVariable("int(I),[520<=I<=590 step 5], int(I2),[I + 5<=I2<=I + 30 step 1]")
```

```
    lngR = udtP.SolveConstraintsOrdered(1)
```

```
    Text1 = Str(lngR)
```

```
End Sub
```

```
15 Private Sub Command1_Click()
```

```
    Text1 = ""
```

```
End Sub
```

```

' frmAbout.frm
VERSION 5.00
Begin VB.Form frmAbout
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "About TCA"
    ClientHeight  = 2610
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 4440
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 2610
    ScaleWidth    = 4440
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
Begin VB.CommandButton cmdOK
    Caption       = "OK"
    Height        = 495
    Left          = 3120
    TabIndex      = 1
    Top           = 120
    Width         = 1215
End
Begin VB.Label lblVersion
    Height        = 255
    Left          = 240
    TabIndex      = 2
    Top           = 2160
    Width         = 2295
End
Begin VB.Label Label1
    Caption       = "TCA is a collaborative development of the Assessment and Research
Divisions."
    Height        = 615
    Left          = 240
    TabIndex      = 0
    Top           = 1320
    Width         = 2535
End
Begin VB.Image imaETS
    BorderStyle   = 1 'Fixed Single
    Height        = 780

```

```

        Left      = 960
        Picture   = "frmAbout.frx":0000
        Top       = 240
        Width     = 1275
5      End
End
Attribute VB_Name = "frmAbout"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
10  Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdEasterEgg_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)

15      If Button = vbRightButton Then
        ' display easter egg
        Beep
        End If

End Sub

20  Private Sub cmdOK_Click()

        Unload Me

End Sub

Private Sub Form_Load()

25      lblVersion = frmSplash.lblVersion

End Sub

Private Sub imaETS_DblClick()

        ' display easter egg
        Beep
30  End Sub

' frmAttributes.frm
VERSION 5.00

```

```

Begin VB.Form frmAttributes
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "Family Attributes"
    ClientHeight  = 1590
5    ClientLeft   = 45
    ClientTop     = 285
    ClientWidth   = 4305
    LinkTopic     = "Form1"
    LockControls  = -1 'True
10    MaxButton    = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 1590
    ScaleWidth    = 4305
    ShowInTaskbar = 0 'False
15    StartUpPosition = 1 'CenterOwner
    Begin VB.ComboBox cboProximity
        Height      = 315
        ItemData     = "frmAttributes.frx":0000
        Left         = 240
20    List        = "frmAttributes.frx":000D
        Style        = 2 'Dropdown List
        TabIndex     = 4
        Top          = 360
        Width        = 1935
25    End
    Begin VB.OptionButton optGeneric
        Caption      = "Generic"
        Height       = 195
        Index        = 0
30    Left        = 120
        TabIndex     = 3
        Top          = 1035
        Value        = -1 'True
        Width        = 975
35    End
    Begin VB.OptionButton optGeneric
        Caption      = "Non-generic"
        Height       = 195
        Index        = 1
40    Left        = 1080
        TabIndex     = 2
        Top          = 1035
        Width        = 1455
    End
45    Begin VB.CommandButton cmdCancel

```

```

Caption      = "Cancel"
Height       = 495
Left         = 3000
TabIndex     = 1
5   ToolTipText = "Click here to return without saving these family attributes."
Top          = 720
Width        = 1215
End
Begin VB.CommandButton cmdOK
10   Caption      = "OK"
      Default     = -1 'True
      Height      = 495
      Left        = 3000
      TabIndex    = 0
15   ToolTipText  = "Click here to save these family attributes."
      Top         = 120
      Width       = 1215
End
Begin VB.Label lbl
20   Caption      = "Variant proximity"
      Height      = 255
      Left        = 240
      TabIndex    = 5
25   Top         = 120
      Width       = 1335
End
End
Attribute VB_Name = "frmAttributes"
Attribute VB_GlobalNameSpace = False
30 Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mblnOK As Boolean

35 Private mblnGeneric As Boolean
Private mudtProximity As Proximity

Private Sub Form_Load()

    mblnOK = False

    cboProximity.ListIndex = frmTCA.Family.Proximity
40   If frmTCA.Family.Generic Then

```

```
    optGeneric(0) = True
Else
    optGeneric(1) = True
End If
```

5

```
    mblnGeneric = frmTCA.Family.Generic
    mudtProximity = frmTCA.Family.Proximity
```

```
End Sub
```

```
Public Property Get Proximity() As Proximity
```

10

```
    Proximity = mudtProximity
```

```
End Property
```

```
Public Property Get Generic() As Boolean
```

```
    Generic = mblnGeneric
```

```
End Property
```

15

```
Private Sub cmdOK_Click()
```

```
    mblnOK = True
```

```
    Unload Me
```

```
End Sub
```

20

```
Private Sub cmdCancel_Click()
```

```
    Unload Me
```

```
End Sub
```

```
Public Property Get OK() As Boolean
```

```
    OK = mblnOK
```

25

```
End Property
```

```
Private Sub cboProximity_Click()
```

```
    mudtProximity = cboProximity.ListIndex
```

End Sub

Private Sub optGeneric_Click(Index As Integer)

 mblnGeneric = optGeneric(0)

End Sub


```

' frmComments.frm
VERSION 5.00
Begin VB.Form frmComments
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "Comments"
    ClientHeight  = 3765
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 5250
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 3765
    ScaleWidth    = 5250
    ShowInTaskbar = 0 'False
    StartUpPosition = 2 'CenterScreen
    Begin VB.CommandButton cmdCancel
        Caption     = "Cancel"
        Height      = 495
        Left        = 3960
        TabIndex    = 2
        ToolTipText  = "Click here to save these family attributes."
        Top         = 720
        Width       = 1215
    End
    Begin VB.CommandButton cmdOK
        Caption     = "OK"
        Default     = -1 'True
        Height      = 495
        Left        = 3960
        TabIndex    = 1
        ToolTipText  = "Click here to save these family attributes."
        Top         = 120
        Width       = 1215
    End
    Begin VB.TextBox txtComment
        Height      = 3495
        Left        = 120
        MultiLine   = -1 'True
        TabIndex    = 0
        Top         = 120
        Width       = 3735
    End
End

```

```
End
Attribute VB_Name = "frmComments"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
5 Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private mstrComment As String
```

```
Public Property Get Comment() As String
```

```
    Comment = mstrComment
```

```
End Property
```

```
Public Property Let Comment(ByVal strNewValue As String)
```

```
    txtComment = strNewValue
    mstrComment = strNewValue
```

```
End Property
```

```
Private Sub cmdCancel_Click()
```

```
    Unload Me
```

```
End Sub
```

```
Private Sub cmdOK_Click()
```

```
    mstrComment = txtComment
    Unload Me
```

```
End Sub
```

```

' frmDifficulty.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Begin VB.Form frmDifficulty
5   BorderStyle   = 4 'Fixed ToolWindow
    ClientHeight  = 8730
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 6855
10   LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 8730
    ScaleWidth    = 6855
15   ShowInTaskbar = 0 'False
    StartUpPosition = 2 'CenterScreen
Begin VB.CheckBox chkRoute
    Caption       = "Route to TCS"
20   Height        = 375
    Left          = 2640
    TabIndex      = 33
    Top           = 1800
    Width         = 1935
25 End
Begin VB.ComboBox cboKey
    Height        = 315
    ItemData      = "frmDifficulty.frx":0000
    Left          = 2640
30   List          = "frmDifficulty.frx":0013
    Style         = 2 'Dropdown List
    TabIndex      = 30
    Top           = 1200
    Width         = 615
35 End
Begin VB.CheckBox chkCalcDifficulty
    Caption       = "Calculate difficulty"
    Height        = 255
    Left          = 240
40   TabIndex      = 27
    Top           = 3600
    Value         = 1 'Checked
    Width         = 1935
End

```

```
Begin VB.ComboBox cboDeliveryMode
    Height      = 315
    ItemData    = "frmDifficulty.frx":0026
    Left        = 2640
    List        = "frmDifficulty.frx":0030
    Style       = 2 'Dropdown List
    TabIndex    = 25
    Top         = 480
    Width       = 1695
```

```
End
```

```
Begin VB.ComboBox cboDomain
    Height      = 315
    ItemData    = "frmDifficulty.frx":003E
    Left        = 240
    List        = "frmDifficulty.frx":004E
    Style       = 2 'Dropdown List
    TabIndex    = 18
    Top         = 1200
    Width       = 1695
```

```
End
```

```
Begin VB.OptionButton optNature
    Caption     = "Pure"
    Height      = 375
    Index       = 0
    Left        = 240
    TabIndex    = 17
    Top         = 1800
    Value       = -1 'True
    Width       = 735
```

```
End
```

```
Begin VB.OptionButton optNature
    Caption     = "Real"
    Height      = 375
    Index       = 1
    Left        = 1200
    TabIndex    = 16
    Top         = 1800
    Width       = 735
```

```
End
```

```
Begin VB.CommandButton cmdOK
    Caption     = "OK"
    Default     = -1 'True
    Height      = 495
    Left        = 5520
    TabIndex    = 8
```

```
ToolTipText = "Click here to save changes and return."  
Top         = 240  
Width       = 1215
```

```
End
```

```
5 Begin VB.CommandButton cmdCancel
```

```
Caption     = "Cancel"  
Height      = 495  
Left        = 5520  
TabIndex    = 7
```

```
10 ToolTipText = "Click here to save changes and return."  
Top         = 840  
Width       = 1215
```

```
End
```

```
Begin VB.TextBox txtBatchId
```

```
15 Height      = 315  
Left        = 240  
TabIndex    = 0  
Top         = 480  
Width       = 1695
```

```
20 End
```

```
Begin ComctlLib.Slider sldTDEstimate
```

```
25 Height      = 375  
Left        = 480  
TabIndex    = 20  
Top         = 2760  
Width       = 3975  
_ExtentX    = 7011  
_ExtentY    = 661  
_Version    = 327682  
30 LargeChange = 1  
Min         = 1  
Max         = 5  
SelStart    = 1  
Value       = 1
```

```
35 End
```

```
Begin VB.Frame fraPredDiff
```

```
Caption     = "Predicted Difficulty"  
Height      = 1575  
Left        = 480  
40 TabIndex    = 10  
Top         = 6720  
Width       = 4575
```

```
Begin ComctlLib.Slider sldDiffEstimate
```

```
45 Height      = 375  
Left        = 240
```

```

        TabIndex      = 11
        Top           = 720
        Width         = 3975
        _ExtentX      = 7011
5       _ExtentY      = 661
        _Version      = 327682
        Min           = 1
        Max            = 5
        SelStart      = 1
10      Value         = 1
End
Begin VB.Label lblIRTValue
    Height      = 255
    Left        = 1080
15    TabIndex   = 32
    Top         = 360
    Width       = 3015
End
Begin VB.Label lblPredEasy
20    Caption    = "Easy"
    Height      = 255
    Left        = 3840
    TabIndex    = 15
    Top         = 1200
25    Width     = 615
End
Begin VB.Label lblPredMed
30    Caption    = "Medium"
    Height      = 255
    Left        = 1920
    TabIndex    = 14
    Top         = 1200
    Width       = 855
End
35 End
Begin VB.Label lblPredDiff
    Caption     = "Difficult"
    Height      = 255
    Left        = 240
    TabIndex    = 13
40    Top         = 1200
    Width       = 735
End
Begin VB.Label lblIRT
45    Caption    = "IRT b:"
    Height      = 255

```

```

        Left      = 360
        TabIndex  = 12
        Top       = 360
        Width     = 495
5      End
End
Begin VB.Frame fraGREDiff
    Caption      = "GRE Difficulty "
    Height       = 4575
10    Left       = 240
    TabIndex     = 2
    Top          = 3960
    Width       = 5055
    Begin VB.ComboBox cboGREConcept
15        Height      = 315
        ItemData     = "frmDifficulty.frx":0080
        Left        = 240
        List        = "frmDifficulty.frx":0093
        Style       = 2 'Dropdown List
20        TabIndex   = 28
        Top         = 2160
        Width      = 2055
    End
    Begin VB.ComboBox cboGRECog
25        Height      = 315
        ItemData     = "frmDifficulty.frx":00ED
        Left        = 240
        List        = "frmDifficulty.frx":00FA
        Style       = 2 'Dropdown List
30        TabIndex   = 5
        Top         = 1440
        Width      = 2055
    End
    Begin VB.ComboBox cboGREComp
35        Height      = 315
        ItemData     = "frmDifficulty.frx":012D
        Left        = 240
        List        = "frmDifficulty.frx":013D
        Style       = 2 'Dropdown List
40        TabIndex   = 3
        Top         = 720
        Width      = 2055
    End
    Begin VB.Label lblConcept
45        Caption     = "Concept:"

```

Height = 255
Left = 240
TabIndex = 29
Top = 1920
Width = 975

End

Begin VB.Label lblGRECog
Caption = "Cognition:"
Height = 255
Left = 240
TabIndex = 6
Top = 1200
Width = 975

End

Begin VB.Label lblGREComp
Caption = "Computation:"
Height = 255
Left = 240
TabIndex = 4
Top = 480
Width = 975

End

End

Begin VB.Frame fraGMATDiff
Caption = "GMAT Difficulty"
Height = 4575
Left = 240
TabIndex = 9
Top = 3960
Width = 5055

End

Begin VB.Frame fraOther
Height = 4575
Left = 240
TabIndex = 34
Top = 3960
Width = 5055

End

Begin VB.Label lblKey
Caption = "Key:"
Height = 255
Left = 2640
TabIndex = 31
Top = 960
Width = 975


```

End
Begin VB.Label lblTarget
    Caption      = "Target template:"
    Height       = 255
5    Left        = 2640
    TabIndex     = 26
    Top          = 240
    Width        = 1815
End
10 Begin VB.Label lblSlideDirections
    Caption      = "Adjust the slide to estimated variant difficulty:"
    Height       = 255
    Left         = 600
    TabIndex     = 24
15    Top         = 2400
    Width        = 3615
End
Begin VB.Label lblTDDiff
    Caption      = "Difficult"
20    Height      = 255
    Left         = 480
    TabIndex     = 23
    Top          = 3240
    Width        = 735
25 End
Begin VB.Label lblTDMed
    Caption      = "Medium"
    Height       = 255
    Left         = 2160
30    TabIndex    = 22
    Top          = 3240
    Width        = 855
End
35 Begin VB.Label lblTDEasy
    Caption      = "Easy"
    Height       = 255
    Left         = 4080
    TabIndex     = 21
40    Top         = 3240
    Width        = 615
End
Begin VB.Label lblDomain
    Caption      = "Domain:"
45    Height      = 255
    Left         = 240

```

```
    TabIndex    = 19
    Top         = 960
    Width       = 975
```

```
End
```

```
Begin VB.Label LblBatch
```

```
    Caption    = "Batch id:"
    Height     = 255
    Left       = 240
    TabIndex   = 1
    Top        = 240
    Width      = 975
```

```
End
```

```
End
```

```
Attribute VB_Name = "frmDifficulty"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = False
```

```
Attribute VB_PredeclaredId = True
```

```
Attribute VB_Exposed = False
```

```
Option Explicit
```

```
Dim mudtFamily As Family
```

```
Dim mudtClone As Clone
```

```
Dim mudtDE As DifficultyEstimate
```

```
Dim mudtGreDE As GREDifficultyEstimate
```

```
Dim mudtGmatDE As GMATDifficultyEstimate
```

```
Dim mblnFormLoad As Boolean
```

```
Public Property Let Family(ByVal udtNewValue As Family)
```

```
    Set mudtFamily = udtNewValue
```

```
End Property
```

```
Public Property Let Clone(ByVal udtNewValue As Clone)
```

```
    Set mudtClone = udtNewValue
```

```
End Property
```

```
Private Sub Form_Load()
```

```
    Set mudtDE = mudtClone.DiffEst
```

```
    mblnFormLoad = True
```

```

' if there's a key, prohibit input.
If mudtFamily.ItemType = ptStandardMC Then
    cboKey.Enabled = False
Else
5    cboKey.Enabled = True
End If

' change form depending on program
Select Case mudtFamily.Program
10    Case prGRE
        fraGREDiff.ZOrder
        fraPredDiff.ZOrder
    Case prGMAT
        fraGMATDiff.ZOrder
15    fraPredDiff.ZOrder
    Case Else
        fraOther.ZOrder
End Select

20    cboDomain.ListIndex = mudtClone.Domain
    txtBatchId = mudtClone.BatchID
    cboDeliveryMode.ListIndex = mudtClone.DeliveryMode

' if key is not set, force "A"
25    If mudtClone.key = "" Then
        cboKey = "A"
    Else
        cboKey = mudtClone.key
    End If

30    If mudtClone.Nature = naPure Then
        optNature(0) = True
    Else
        optNature(1) = True
35    End If

    sldTDEstimate = mudtClone.TDEstimate
    chkRoute = mudtClone.IsRouted
    chkCalcDifficulty = mudtClone.IsDifficultyCalculated
40    chkCalcDifficulty_Click ' update screen accordingly
    If mudtClone.IsDifficultyCalculated Then
        Select Case mudtFamily.Program
            Case prGRE
                Set mudtGreDE = mudtClone.DiffEst
45                cboGREComp.ListIndex = mudtGreDE.Computation

```

```

        cboGRECog.ListIndex = mudtGreDE.Cognition
        cboGREConcept.ListIndex = mudtGreDE.Concept
        CreateDiffEst
    Case prGMAT
5       Set mudtGmatDE = mudtClone.DiffEst
        ' nothing to load
        CreateDiffEst
    Case prSAT
        ' do nothing
10      End Select
    Else
        cboGREComp.ListIndex = 0
        cboGRECog.ListIndex = 0
        cboGREConcept.ListIndex = 0
15      End If

        mblnFormLoad = False

End Sub

20 Private Sub cmdOK_Click()

    CreateProfile

    Unload Me

25 End Sub

Private Sub cmdCancel_Click()

    Unload Me

End Sub

30 Private Sub cboDomain_Click()

    CreateProfile

End Sub

Private Sub cboGRECog_Click()

    CreateProfile

35 End Sub

```

Private Sub cboGREComp_Click()

 CreateProfile

End Sub

5 Private Sub cboGREConcept_Click()

 CreateProfile

End Sub

Private Sub cboKey_Click()

10 CreateProfile

End Sub

Private Sub optNature_Click(Index As Integer)

 CreateProfile

End Sub

Private Sub sldTDEstimate_Click()

 CreateProfile

End Sub

Private Sub chkCalcDifficulty_Click()

 fraPredDiff.Enabled = CBool(chkCalcDifficulty)

 fraGREDiff.Enabled = CBool(chkCalcDifficulty)

 fraGMATDiff.Enabled = CBool(chkCalcDifficulty)

25 lblGREComp.Enabled = CBool(chkCalcDifficulty)

 cboGREComp.Enabled = CBool(chkCalcDifficulty)

 lblGRECog.Enabled = CBool(chkCalcDifficulty)

 cboGRECog.Enabled = CBool(chkCalcDifficulty)

 lblConcept.Enabled = CBool(chkCalcDifficulty)

30 cboGREConcept.Enabled = CBool(chkCalcDifficulty)

 lblIRT.Enabled = CBool(chkCalcDifficulty)

 lblIRTValue.Enabled = CBool(chkCalcDifficulty)

 lblPredDiff.Enabled = CBool(chkCalcDifficulty)

 lblPredEasy.Enabled = CBool(chkCalcDifficulty)

```
lblPredMed.Enabled = CBool(chkCalcDifficulty)
lblPredDiff.Enabled = CBool(chkCalcDifficulty)
```

```
5 If chkCalcDifficulty Then
    CreateProfile
End If
```

```
End Sub
```

```
Private Sub CreateProfile()
```

```
10 ' don't do it if were still loading form
    If mbInFormLoad Then Exit Sub
```

```
15 mudtClone.Program = mudtFamily.Program
    mudtClone.Domain = cboDomain.ListIndex
    mudtClone.BatchID = txtBatchId
    mudtClone.DeliveryMode = cboDeliveryMode.ListIndex
    mudtClone.key = cboKey
    If optNature(0) = True Then
        mudtClone.Nature = naPure
    Else
        mudtClone.Nature = naReal
    End If
    mudtClone.IsRouted = chkRoute
    mudtClone.TDEstimate = sldTDEstimate
```

```
    mudtClone.IsDifficultyCalculated = chkCalcDifficulty
```

```
25 If chkCalcDifficulty Then
    CreateDiffEst
End If
```

```
End Sub
```

```
Private Sub CreateDiffEst()
```

```
30 If mudtClone.IsDifficultyCalculated Then
    Set mudtDE = Nothing
    Select Case mudtFamily.Program
        Case prGRE
            Set mudtGreDE = Nothing
35 Set mudtGreDE = New GREDifficultyEstimate
            mudtGreDE.Domain = cboDomain.ListIndex
            mudtGreDE.Computation = cboGREComp.ListIndex
```

```

    mudtGreDE.Cognition = cboGRECog.ListIndex
    mudtGreDE.Concept = cboGREConcept.ListIndex
    mudtGreDE.key = cboKey
    If optNature(0) = True Then
        mudtGreDE.Nature = naPure
    Else
        mudtGreDE.Nature = naReal
    End If
    mudtGreDE.ItemType = mudtFamily.ItemType
    ' attach this GRE DE to the clone
    mudtClone.DiffEst = mudtGreDE
    Set mudtDE = mudtGreDE
    SetPredDiffSlider
    Case prGMAT
        Set mudtGmatDE = Nothing
        Set mudtGmatDE = New GMATDifficultyEstimate
        mudtGmatDE.Domain = cboDomain.ListIndex
        mudtGmatDE.key = cboKey
        If optNature(0) = True Then
            mudtGmatDE.Nature = naPure
        Else
            mudtGmatDE.Nature = naReal
        End If
        mudtGmatDE.ItemType = mudtFamily.ItemType
        mudtGmatDE.TDDiffEst = sldTDEstimate
        ' attach this GMAT DE to the clone
        mudtClone.DiffEst = mudtGmatDE
        Set mudtDE = mudtGmatDE
        SetPredDiffSlider
    Case prSAT
        ' do nothing
    End Select
Else ' opted not to calc difficulty
    mudtClone.DiffEst = Nothing
End If

End Sub

Private Sub SetPredDiffSlider()

    Dim dblIIRT As Double

    dblIIRT = mudtDE.ComputeDifficulty

    lblIIRTValue = Format(dblIIRT, "0.#")

```

Select Case mudtFamily.Program

Case prGRE

If dblIRT < -1.001 Then

sldDiffEstimate = 5

ElseIf dblIRT < -0.238 Then

sldDiffEstimate = 4

ElseIf dblIRT < 0.379 Then

sldDiffEstimate = 3

ElseIf dblIRT < 0.931 Then

sldDiffEstimate = 2

Else

sldDiffEstimate = 1

End If

Case prGMAT

If dblIRT < -0.919 Then

sldDiffEstimate = 5

ElseIf dblIRT < -0.093 Then

sldDiffEstimate = 4

ElseIf dblIRT < 0.565 Then

sldDiffEstimate = 3

ElseIf dblIRT < 1.197 Then

sldDiffEstimate = 2

Else

sldDiffEstimate = 1

End If

End Select

End Sub


```

' frmDrag.frm
VERSION 5.00
Begin VB.Form frmDrag
    Caption       = "Window drag control"
5    ClientHeight = 1005
    ClientLeft    = 60
    ClientTop     = 345
    ClientWidth   = 3060
    LinkTopic     = "Form1"
10    ScaleHeight  = 1005
    ScaleWidth    = 3060
    StartupPosition = 2 'CenterScreen
    Begin VB.CommandButton Command2
        Caption     = "Full Drag OFF"
15        Height      = 735
        Left         = 1560
        TabIndex     = 1
        Top          = 120
        Width        = 1215
20    End
    Begin VB.CommandButton Command1
        Caption     = "Full Drag ON"
        Height      = 735
        Left         = 120
25        TabIndex     = 0
        Top          = 120
        Width        = 1215
    End
30    End
    Attribute VB_Name = "frmDrag"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = True
    Attribute VB_Exposed = False
35    Option Explicit

    Private Declare Function SystemParametersInfo Lib "user32" _
        Alias "SystemParametersInfoA" (ByVal uAction As Long, _
        ByVal uParam As Long, ByRef lpvParam As Any, _
        ByVal fuWinIni As Long) As Long
40

    Private Const SPI_GETDRAGFULLWINDOWS = 38
    Private Const SPI_SETDRAGFULLWINDOWS = 37
    Private Const SPIF_SENDWININICHANGE = 2

```

```
Public Function IsFullWindowDragOn() As Boolean
```

```
    Dim result As Long
```

```
    'Call API and check for successful call.
```

```
    If SystemParametersInfo(SPI_GETDRAGFULLWINDOWS, 0&, result, 0&) <> 0 Then
```

```
        'Feature supported now check value of result.
```

```
        If result = 0 Then
```

```
            IsFullWindowDragOn = False
```

```
        Else
```

```
            IsFullWindowDragOn = True
```

```
        End If
```

```
        'Call failed, feature not supported.
```

```
    Else
```

```
        IsFullWindowDragOn = False
```

```
    End If
```

```
End Function
```

```
Private Sub TurnOffFullWindowDrag()
```

```
    Dim result As Long
```

```
    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 0&, _  
        ByVal vbNullString, SPIF_SENDWININICHANGE)
```

```
End Sub
```

```
Private Sub TurnOnFullWindowDrag()
```

```
    Dim result As Long
```

```
    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 1&, _  
        ByVal vbNullString, SPIF_SENDWININICHANGE)
```

```
End Sub
```

```
Private Sub Command1_Click()
```

```
    TurnOnFullWindowDrag
```

```
End Sub
```

```
Private Sub Command2_Click()
```

TurnOffFullWindowDrag

End Sub

```

' frmIED.frm
VERSION 5.00
Begin VB.Form frmIED
    BorderStyle   = 1 'Fixed Single
    Caption       = "TCA Installation"
    ClientHeight  = 1185
    ClientLeft    = 45
    ClientTop     = 330
    ClientWidth   = 2475
    LinkTopic     = "Form1"
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 1185
    ScaleWidth    = 2475
    StartUpPosition = 2 'CenterScreen
    Begin VB.CommandButton cmdOK
        Caption    = "OK"
        Height     = 375
        Left       = 600
        TabIndex   = 1
        Top        = 720
        Width      = 1215
    End
    Begin VB.Label Label1
        Caption    = "Setting IED files to read-only."
        Height     = 255
        Left      = 240
        TabIndex   = 0
        Top       = 240
        Width     = 2055
    End
End
Attribute VB_Name = "frmIED"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdOK_Click()

    Unload Me

End Sub

```

Private Sub Form_Load()

Call Shell("attrib +r C:\tcs\working\dscbt.ied", vbHide)

Call Shell("attrib +r C:\tcs\working\qccbt.ied", vbHide)

Call Shell("attrib +r C:\tcs\working\qcppt.ied", vbHide)

5 Call Shell("attrib +r C:\tcs\working\ssmccbt.ied", vbHide)

Call Shell("attrib +r C:\tcs\working\ssmcppt.ied", vbHide)

End Sub

VBSCA -80-

```

' frmIndexedString.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Begin VB.Form frmIndexedString
5   BorderStyle   = 4 'Fixed ToolWindow
    ClientHeight  = 2265
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 5835
10   LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 2265
15   ScaleWidth    = 5835
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
Begin ComctlLib.ListView lvwIndexed
    Height        = 1815
20   Left         = 120
    TabIndex      = 6
    Top           = 120
    Width         = 4215
    _ExtentX      = 7435
25   _ExtentY     = 3201
    View          = 3
    Arrange       = 2
    LabelEdit     = 1
    MultiSelect   = -1 'True
30   LabelWrap    = -1 'True
    HideSelection = 0 'False
    _Version      = 327682
    ForeColor     = -2147483640
    BackColor     = -2147483643
35   BorderStyle  = 1
    Appearance    = 1
    NumItems      = 2
BeginProperty ColumnHeader(1) {0713E8C7-850A-101B-AFC0-4210102A8DA7}
    Key           = ""
40   Object.Tag    = ""
    Text          = "Index"
    Object.Width   = 529
EndProperty
BeginProperty ColumnHeader(2) {0713E8C7-850A-101B-AFC0-4210102A8DA7}

```

```

        SubItemIndex = 1
        Key          = ""
        Object.Tag    = ""
        Text         = "Value"
5       Object.Width = 6174
    EndProperty
End
Begin VB.CommandButton cmdAdd
    Caption      = "Add"
10   Height     = 255
    Left        = 120
    TabIndex     = 5
    ToolTipText  = "Click here to add a value to the end of the list."
    Top         = 1900
15   Width      = 975
End
Begin VB.CommandButton cmdInsert
    Caption      = "Insert"
20   Height     = 255
    Left        = 1080
    TabIndex     = 4
    ToolTipText  = "Click here to insert a value before the currently selected value."
    Top         = 1900
25   Width      = 1095
End
Begin VB.CommandButton cmdEdit
    Caption      = "Edit"
30   Height     = 255
    Left        = 2160
    TabIndex     = 3
    ToolTipText  = "Click here to edit the currently selected value."
    Top         = 1900
    Width       = 1095
End
35 Begin VB.CommandButton cmdRemove
    Caption      = "Remove"
    Height       = 255
    Left        = 3240
    TabIndex     = 2
40   ToolTipText = "Click here to remove the selected value."
    Top         = 1900
    Width       = 1095
End
45 Begin VB.CommandButton cmdStrOK
    Caption      = "OK"

```

```

        Default      = -1 'True
        Height       = 495
        Left         = 4440
        TabIndex     = 0
5       ToolTipText  = "Click here to save changes and return."
        Top         = 120
        Width       = 1215
    End
    Begin VB.CommandButton cmdStrCancel
10       Caption      = "Cancel"
        Height       = 495
        Left         = 4440
        TabIndex     = 1
        ToolTipText  = "Click here to return without saving changes."
15       Top         = 720
        Width       = 1215
    End
    Begin VB.Menu mnuIndexed
        Caption      = "Indexed"
20       Visible     = 0 'False
        Begin VB.Menu mnuIndexedAdd
            Caption    = "Add"
        End
        Begin VB.Menu mnuIndexedInsert
25       Caption     = "Insert"
        End
        Begin VB.Menu mnuIndexedEdit
            Caption    = "Edit"
        End
30       Begin VB.Menu mnuIndexedRemove
            Caption    = "Remove"
        End
    End
End
35 Attribute VB_Name = "frmIndexedString"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
40 Option Explicit

Private mudtModel As Model
Private mudtEF As EditFlags
Private mstrVariableName As String
Private mcolStrings As Collection

```


Private mblnOK As Boolean

Public Property Let Model(ByVal udtNewValue As Model)

Set mudtModel = udtNewValue

End Property

5 Public Property Let AddEditFlag(ByVal udtNewValue As EditFlags)

mudtEF = udtNewValue

End Property

Public Property Let SubStringCollection(ByVal colNewValue As Collection)

10 Set mcolStrings = colNewValue

End Property

Private Sub cmdAdd_Click()

Call mnuIndexedAdd_Click

End Sub

Private Sub cmdEdit_Click()

Call mnuIndexedEdit_Click

End Sub

Private Sub cmdInsert_Click()

Call mnuIndexedInsert_Click

End Sub

25 Private Sub cmdRemove_Click()

Call mnuIndexedRemove_Click

End Sub

Private Sub Form_Load()

```
Dim varS As Variant
Dim lsiLI As ListItem
```

```
Dim udtWAPI As New Win32API
```

```
5 ' enable full row select
Call udtWAPI.EnableListViewFullRowSelect(lvwIndexed)
```

```
mblnOK = False
```

```
frmIndexedString.Caption = "Editing substrings of string " & mstrVariableName
```

```
10 If mudtEF = aeEdit Then
    With lvwIndexed
        For Each varS In mcolStrings
            Set lsiLI = .ListItems.Add
            UpdateListView
15         lsiLI.SubItems(1) = varS
        Next varS
    End With
End If
```

```
20 ' prevent changes if model is frozen
If mudtModel.IsFrozen Then
    cmdStrOK.Enabled = False
    cmdAdd.Enabled = False
    mnuIndexedAdd.Enabled = False
25    cmdEdit.Caption = "Browse"
    mnuIndexedEdit.Caption = "Browse"
    cmdInsert.Enabled = False
    mnuIndexedInsert.Enabled = False
    cmdRemove.Enabled = False
30    mnuIndexedRemove.Enabled = False
End If
```

```
End Sub
```

```
Public Property Let VariableName(ByVal strNewValue As String)
```

```
35     mstrVariableName = strNewValue
```

```
End Property
```

```
Public Property Get StringValue() As String
```

Dim udtSS As New SubString

udtSS.Delimiter = Chr(STRING_DELIMITER)

udtSS.StringCollection = mcolStrings

StringValue = udtSS.StringValue

End Property

Public Property Get SubStringCollection() As Collection

Set SubStringCollection = mcolStrings

End Property

Public Property Get OK() As Boolean

OK = mblnOK

End Property

Private Sub cmdStrOK_Click()

Dim lsiItem As ListItem

Set mcolStrings = New Collection

For Each lsiItem In lvwIndexed.ListItems

Call mcolStrings.Add(lsiItem.SubItems(1))

Next lsiItem

mblnOK = True

Unload Me

End Sub

Private Sub cmdStrCancel_Click()

Unload Me

End Sub

Private Sub mnuIndexedAdd_Click()

With frmString

```

' set the model
.Model = mudtModel
' set the string
.StringValue = ""
5 ' set var name
.VariableName = mstrVariableName & "." _
    & Trim(Str(lvwIndexed.ListItems.Count + 1))
' do it
.Show vbModal
10 If .OK = False Then Exit Sub
End With

```

```

Dim lsiNewItem As ListItem

```

```

Set lsiNewItem = lvwIndexed.ListItems.Add
15 UpdateListView
lsiNewItem.SubItems(1) = frmString.StringValue

```

```

End Sub

```

```

Private Sub mnuIndexedEdit_Click()

```

```

20 With frmString
    ' set the model
    .Model = mudtModel
    ' set the string
    .StringValue = lvwIndexed.SelectedItem.SubItems(1)
25 ' set var name
    .VariableName = mstrVariableName & "." _
        & Trim(Str(lvwIndexed.SelectedItem.Index))
    ' do it
    .Show vbModal
30 If .OK = False Then Exit Sub
End With

```

```

lvwIndexed.SelectedItem.SubItems(1) = frmString.StringValue

```

```

End Sub

```

```

Private Sub mnuIndexedInsert_Click()

```

```

35 If lvwIndexed.SelectedItem Is Nothing Then Exit Sub

```

```

With frmString
    ' set the Model

```

```

        .Model = mudtModel
        ' set the string
        .StringValue = ""
        ' set var name
5      .VariableName = mstrVariableName
        ' do it
        .Show vbModal
        If .OK = False Then Exit Sub
    End With

10    Dim lsiNewItem As ListItem

        Set lsiNewItem = lvwIndexed.ListItems.Add(lvwIndexed.SelectedItem.Index)
        UpdateListView
        lsiNewItem.SubItems(1) = frmString.StringValue

15    End Sub

Private Sub mnuIndexedRemove_Click()

    If lvwIndexed.SelectedItem Is Nothing Then Exit Sub

20    Call lvwIndexed.ListItems.Remove(lvwIndexed.SelectedItem.Index)
        UpdateListView

    End Sub

Private Sub UpdateListView()

25    Dim intI As Integer

        For intI = 1 To lvwIndexed.ListItems.Count
            lvwIndexed.ListItems.Item(intI).Text = Str(intI)
        Next intI

30    End Sub

```

```

' frmNew.frm
VERSION 5.00
Begin VB.Form frmNew
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "New family properties"
    ClientHeight  = 1740
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 6240
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 1740
    ScaleWidth    = 6240
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
Begin VB.CommandButton cmdCancel
    Cancel        = -1 'True
    Caption       = "Cancel"
    Height        = 495
    Left          = 4800
    TabIndex      = 9
    Top           = 720
    Width         = 1215
End
Begin VB.CommandButton cmdOK
    Caption       = "OK"
    Default       = -1 'True
    Height        = 495
    Left          = 4800
    TabIndex      = 8
    Top           = 120
    Width         = 1215
End
Begin VB.OptionButton optGeneric
    Caption       = "Non-generic"
    Height        = 195
    Index         = 1
    Left          = 3240
    TabIndex      = 7
    Top           = 1150
    Width         = 1455
End

```

```

Begin VB.OptionButton optGeneric
  Caption      = "Generic"
  Height       = 195
  Index        = 0
  Left         = 2280
  TabIndex     = 6
  Top          = 1150
  Value        = -1 'True
  Width        = 975
End
Begin VB.ComboBox cboProximity
  Height       = 315
  ItemData     = "frmNew.frx":0000
  Left         = 2280
  List         = "frmNew.frx":000D
  Style        = 2 'Dropdown List
  TabIndex     = 4
  Top          = 360
  Width        = 1935
End
Begin VB.ComboBox cboItemType
  Height       = 315
  ItemData     = "frmNew.frx":0024
  Left         = 120
  List         = "frmNew.frx":0031
  Style        = 2 'Dropdown List
  TabIndex     = 2
  Top          = 1080
  Width        = 1935
End
Begin VB.ComboBox cboProgram
  Height       = 315
  ItemData     = "frmNew.frx":0072
  Left         = 120
  List         = "frmNew.frx":007F
  Style        = 2 'Dropdown List
  TabIndex     = 0
  Top          = 360
  Width        = 1935
End
Begin VB.Label lbl
  Caption      = "Variant proximity"
  Height       = 255
  Left         = 2280
  TabIndex     = 5

```

```

        Top      = 120
        Width    = 1335
    End
    Begin VB.Label lblItemType
5       Caption   = "Item type"
        Height    = 255
        Left      = 120
        TabIndex  = 3
        Top       = 840
10      Width     = 1335
    End
    Begin VB.Label lblProgram
        Caption    = "Program"
        Height     = 255
15      Left      = 120
        TabIndex   = 1
        Top        = 120
        Width      = 1335
    End
20  End
    Attribute VB_Name = "frmNew"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = True
25  Attribute VB_Exposed = False
    Option Explicit

    Private mblnOK As Boolean

    Private mudtProgram As Program
    Private mudtItemType As ItemType
30  Private mudtProximity As Proximity
    Private mblnGeneric As Boolean

    Private Sub Form_Load()

        mblnOK = False

35      ' init combo boxes
        cboProgram.ListIndex = 0
        cboItemType.ListIndex = 0
        cboProximity.ListIndex = 0

40  End Sub

```


Public Property Get OK() As Boolean

OK = mblnOK

End Property

5 Public Property Get Program() As Program

Program = mudtProgram

End Property

Public Property Get ItemType() As ItemType

ItemType = mudtItemType

10

End Property

Public Property Get Proximity() As Proximity

Proximity = mudtProximity

End Property

15 Public Property Get Generic() As Boolean

Generic = mblnGeneric

End Property

Private Sub cboProgram_Click()

mudtProgram = cboProgram.ListIndex

20 End Sub

Private Sub cboItemType_Click()

mudtItemType = cboItemType.ListIndex

End Sub

Private Sub cboProximity_Click()

25 mudtProximity = cboProximity.ListIndex

End Sub

Private Sub optGeneric_Click(Index As Integer)

 mblnGeneric = optGeneric(0)

End Sub

5 Private Sub cmdOK_Click()

 mblnOK = True

 Unload Me

End Sub

10 Private Sub cmdCancel_Click()

 Unload Me

End Sub

' frmNewModel.frm

VERSION 5.00

Begin VB.Form frmNewFamily

BorderStyle = 4 'Fixed ToolWindow

Caption = "New family"

ClientHeight = 1350

ClientLeft = 45

ClientTop = 285

ClientWidth = 4680

LinkTopic = "Form1"

LockControls = -1 'True

MaxButton = 0 'False

MinButton = 0 'False

ScaleHeight = 1350

ScaleWidth = 4680

ShowInTaskbar = 0 'False

StartPosition = 1 'CenterOwner

Begin VB.OptionButton optModelType

Caption = "Quantitative Comparision"

Height = 255

Index = 1

Left = 480

TabIndex = 4

Top = 480

Width = 2535

End

Begin VB.OptionButton optModelType

Caption = "Data Sufficiency"

Height = 255

Index = 2

Left = 480

TabIndex = 3

Top = 720

Width = 2535

End

Begin VB.OptionButton optModelType

Caption = "Standard Multiple Choice"

Height = 255

Index = 0

Left = 480

TabIndex = 2

Top = 240

Value = -1 'True

Width = 2535

```

End
Begin VB.CommandButton cmdCancel
    Caption      = "Cancel"
    Height       = 495
5    Left        = 3360
    TabIndex     = 1
    ToolTipText  = "Click here to return without opening creating a new model."
    Top          = 720
    Width        = 1215

```

```

10 End
Begin VB.CommandButton cmdNewCreate
    Caption      = "Create"
    Default      = -1 'True
    Height       = 495
15    Left        = 3360
    TabIndex     = 0
    ToolTipText  = "Click here to create the new family."
    Top          = 120
    Width        = 1215

```

```

20 End
End
Attribute VB_Name = "frmNewFamily"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
25 Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

```

```

Private mblnOK As Boolean

```

```

' holds the item type

```

```

30 Private mudtItemType As ItemType

```

```

Public Property Get OK() As Boolean

```

```

    OK = mblnOK

```

```

End Property

```

```

35 Public Property Get ItemType() As ItemType

```

```

    ItemType = mudtItemType

```

```

End Property

```

```
Private Sub cmdNewCreate_Click()
```

```
    mblnOK = True
```

```
5    Unload Me
```

```
End Sub
```

```
Private Sub cmdCancel_Click()
```

```
    mblnOK = False
```

```
10    Unload Me
```

```
End Sub
```

```
Private Sub optModelType_Click(Index As Integer)
```

```
    mudtItemType = Index
```

```
End Sub
```

```

' frmProgram.frm
VERSION 5.00
Begin VB.Form frmProgram
    Caption       = "Select the program"
    ClientHeight  = 1350
    ClientLeft    = 60
    ClientTop     = 345
    ClientWidth   = 3225
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    ScaleHeight   = 1350
    ScaleWidth    = 3225
    StartUpPosition = 1 'CenterOwner
Begin VB.OptionButton optProgram
    Caption       = "SAT"
    Height        = 195
    Index         = 2
    Left          = 240
    TabIndex      = 4
    Top           = 720
    Width         = 1335
End
Begin VB.OptionButton optProgram
    Caption       = "GMAT"
    Height        = 195
    Index         = 1
    Left          = 240
    TabIndex      = 3
    Top           = 480
    Width         = 1335
End
Begin VB.OptionButton optProgram
    Caption       = "GRE"
    Height        = 195
    Index         = 0
    Left          = 240
    TabIndex      = 2
    Top           = 240
    Value         = -1 'True
    Width         = 1335
End
Begin VB.CommandButton cmdCancel
    Caption       = "Cancel"
    Height        = 495

```

```

    Left      = 1920
    TabIndex  = 1
    ToolTipText = "Click here to return."
    Top       = 720
5    Width    = 1215
End
Begin VB.CommandButton cmdOK
    Caption    = "OK"
    Height     = 495
10   Left     = 1920
    TabIndex   = 0
    ToolTipText = "Click here to save the currently selected program and return."
    Top        = 120
    Width      = 1215
15   End
End
Attribute VB_Name = "frmProgram"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
20 Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private mblnOK As Boolean

Private mudtProgram As Program

25 Public Property Get OK() As Boolean

    OK = mblnOK

End Property

Public Property Get Program() As Program

30     Program = mudtProgram

End Property

Private Sub cmdOK_Click()

35     mblnOK = True

    Unload Me

```

End Sub

Private Sub cmdCancel_Click()

 mblnOK = False

 Unload Me

5 End Sub

Private Sub optProgram_Click(Index As Integer)

 mudtProgram = Index

End Sub

VBSCA -99-


```

' frmProgress.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.2#0"; "COMCTL32.OCX"
Begin VB.Form frmProgress
5   BorderStyle   = 1 'Fixed Single
    ClientHeight  = 1110
    ClientLeft    = 15
    ClientTop     = 15
    ClientWidth   = 4500
10   ClipControls = 0 'False
    ControlBox    = 0 'False
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
15   MinButton     = 0 'False
    ScaleHeight   = 1110
    ScaleWidth    = 4500
    StartupPosition = 2 'CenterScreen
Begin ComctlLib.ProgressBar prbProgressBar
20   Height       = 255
    Left          = 240
    TabIndex      = 0
    Top           = 600
    Width         = 3975
25   _ExtentX     = 7011
    _ExtentY     = 450
    _Version      = 327682
    Appearance    = 1
    Max           = 500
30 End
Begin VB.Label lblProgress
    Alignment      = 2 'Center
    BeginProperty Font
        Name       = "MS Sans Serif"
35        Size      = 8.25
        Charset    = 0
        Weight     = 700
        Underline   = 0 'False
        Italic      = 0 'False
40        Strikethrough = 0 'False
    EndProperty
    Height        = 255
    Left          = 240
    TabIndex      = 1

```

```
Top      = 240
Width    = 3855
```

```
End
```

```
End
```

```
5  Attribute VB_Name = "frmProgress"
   Attribute VB_GlobalNameSpace = False
   Attribute VB_Creatable = False
   Attribute VB_PredeclaredId = True
   Attribute VB_Exposed = False
10 Option Explicit
```

```

' frmProlog.frm
VERSION 5.00
Begin VB.Form frmProlog
    BorderStyle   = 5 'Sizable ToolWindow
5    ClientHeight = 900
    ClientLeft    = 2775
    ClientTop     = 3720
    ClientWidth   = 4440
    LinkTopic     = "Form1"
10    LockControls = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 900
    ScaleWidth    = 4440
15    ShowInTaskbar = 0 'False
    StartUpPosition = 2 'CenterScreen
    Begin VB.CommandButton cmdAbort
        Caption    = "Abort"
        Default    = -1 'True
20        Height    = 495
        Left       = 3120
        TabIndex   = 0
        Top        = 120
        Width      = 1215
25    End
    Begin VB.Label lblProlog
        Height      = 495
        Left        = 120
        TabIndex    = 1
30        Top       = 120
        Width       = 2655
    End
End
Attribute VB_Name = "frmProlog"
35 Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Option Explicit

40 Private mblnAbort As Boolean

Public Property Get Abort() As Boolean

```

Abort = mblnAbort

End Property

Public Sub Kill()

5 Unload Me

End Sub

Private Sub Form_Load()

 mblnAbort = False

10 End Sub

Private Sub cmdAbort_Click()

 mblnAbort = True

 Unload Me

End Sub

```

' frmSplash.frm
VERSION 5.00
Begin VB.Form frmSplash
    BorderStyle   = 3 'Fixed Dialog
    ClientHeight  = 4245
    ClientLeft    = 255
    ClientTop     = 1410
    ClientWidth   = 7380
    ClipControls  = 0 'False
    ControlBox    = 0 'False
    Icon          = "frmSplash.frx":0000
    KeyPreview    = -1 'True
    LinkTopic     = "Form2"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 4245
    ScaleWidth    = 7380
    ShowInTaskbar = 0 'False
    StartUpPosition = 2 'CenterScreen
Begin VB.Frame fraSplash
    Height        = 4050
    Left          = 120
    TabIndex      = 0
    Top           = 60
    Width         = 7080
Begin VB.Image imgLogo
    BorderStyle   = 1 'Fixed Single
    Height        = 780
    Left          = 600
    Picture       = "frmSplash.frx":000C
    Top           = 720
    Width         = 1275
End
Begin VB.Label lblCopyright
    Caption       = "Copyright 1999"
BeginProperty Font
    Name          = "Arial"
    Size          = 8.25
    Charset       = 0
    Weight        = 400
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False

```

```

EndProperty
Height      = 255
Left       = 4560
TabIndex   = 3
Top        = 3480
Width      = 2415
End
Begin VB.Label lblCompany
Caption     = "Educational Testing Service"
BeginProperty Font
    Name      = "Arial"
    Size      = 8.25
    Charset   = 0
    Weight    = 400
    Underline  = 0 'False
    Italic    = 0 'False
    Strikethrough = 0 'False
EndProperty
Height      = 255
Left       = 4560
TabIndex   = 2
Top        = 3720
Width      = 2415
End
Begin VB.Label lblWarning
Caption     = "Proprietary and Confidential"
BeginProperty Font
    Name      = "Arial"
    Size      = 9.75
    Charset   = 0
    Weight    = 700
    Underline  = 0 'False
    Italic    = 0 'False
    Strikethrough = 0 'False
EndProperty
Height      = 315
Left       = 240
TabIndex   = 1
Top        = 3600
Width      = 2775
End
Begin VB.Label lblVersion
Alignment   = 1 'Right Justify
AutoSize    = -1 'True
Caption     = "Version 1.25"

```

```
BeginProperty Font
  Name      = "Arial"
  Size      = 12
  Charset   = 0
  Weight    = 700
  Underline  = 0 'False
  Italic     = 0 'False
  Strikethrough = 0 'False
```

```
EndProperty
Height      = 285
Left        = 5265
TabIndex    = 4
Top         = 2880
Width       = 1410
```

```
End
Begin VB.Label lblProductName
```

```
  AutoSize   = -1 'True
  Caption    = "Assistant"
  BeginProperty Font
    Name      = "Arial"
    Size      = 48
    Charset   = 0
    Weight    = 700
    Underline  = 0 'False
    Italic     = 0 'False
    Strikethrough = 0 'False
```

```
  EndProperty
  Height     = 1125
  Left       = 1440
  TabIndex   = 6
  Top        = 1560
  Width      = 4320
```

```
End
Begin VB.Label lblCompanyProduct
```

```
  AutoSize   = -1 'True
  Caption    = "Test Creation "
  BeginProperty Font
    Name      = "Arial"
    Size      = 18
    Charset   = 0
    Weight    = 700
    Underline  = 0 'False
    Italic     = 0 'False
    Strikethrough = 0 'False
```

```
  EndProperty
```

```

        Height    = 435
        Left      = 2400
        TabIndex  = 5
        Top       = 1080
5      Width     = 2400
      End
    End
  End
  Attribute VB_Name = "frmSplash"
10  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = True
  Attribute VB_Exposed = False

  Option Explicit

15  Public Sub UnloadMe()

      Unload Me

  End Sub

```



```

' SetPrecision.frm
VERSION 5.00
Begin VB.Form frmSetPrecision
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "Set Precision"
    ClientHeight  = 1965
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 3540
    LinkTopic     = "Form1"
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 1965
    ScaleWidth    = 3540
    ShowInTaskbar = 0 'False
    StartUpPosition = 2 'CenterScreen
Begin VB.CommandButton cmdSetPrecisionDefault
    Caption       = "Default"
    Height        = 495
    Left          = 2160
    TabIndex      = 3
    ToolTipText   = "Click here to return to the default value for precision."
    Top           = 1320
    Width         = 1215
End
Begin VB.CommandButton cmdSetPrecisionOK
    Caption       = "OK"
    Default       = -1 'True
    Height        = 495
    Left          = 2160
    TabIndex      = 2
    ToolTipText   = "Click here to save the displayed value."
    Top           = 120
    Width         = 1215
End
Begin VB.CommandButton cmdSetPrecisionCancel
    Caption       = "Cancel"
    Height        = 495
    Left          = 2160
    TabIndex      = 1
    ToolTipText   = "Click here to return without saving any changes to precision."
    Top           = 720
    Width         = 1215
End

```

```

Begin VB.TextBox txtPrecision
    Height      = 315
    Left        = 120
    TabIndex    = 0
5    Text       = ".1"
    Top         = 120
    Width       = 1815
End
End
10 Attribute VB_Name = "frmSetPrecision"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
15 Option Explicit

Private Sub cmdSetPrecisionCancel_Click()

    Unload Me

End Sub

20 Private Sub cmdSetPrecisionDefault_Click()

    txtPrecision = ".001"

End Sub

Private Sub cmdSetPrecisionOK_Click()
25     frmTCA.Precision = txtPrecision
    Unload Me

End Sub

Private Sub Form_Load()

30     txtPrecision = frmTCA.Precision

End Sub

Private Sub txtPrecision_GotFocus()

35     ' Automatically select all text when TextBox gets focus
    Call txtSelectAll(txtPrecision)

```

End Sub

```

' String.frm
VERSION 5.00
Begin VB.Form frmString
    BorderStyle   = 4 'Fixed ToolWindow
    ClientHeight  = 2265
    ClientLeft    = 45
    ClientTop     = 285
    ClientWidth   = 5835
    LinkTopic     = "Form1"
    LockControls  = -1 'True
    MaxButton     = 0 'False
    MinButton     = 0 'False
    ScaleHeight   = 2265
    ScaleWidth    = 5835
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
    Begin VB.CommandButton cmdStrOK
        Caption     = "OK"
        Default     = -1 'True
        Height      = 495
        Left        = 4440
        TabIndex    = 1
        ToolTipText  = "Click here to save changes and return."
        Top         = 120
        Width       = 1215
    End
    Begin VB.CommandButton cmdStrCancel
        Caption     = "Cancel"
        Height      = 495
        Left        = 4440
        TabIndex    = 2
        ToolTipText  = "Click here to return without saving changes."
        Top         = 720
        Width       = 1215
    End
    Begin VB.TextBox txtString
        Height      = 315
        Left        = 240
        TabIndex    = 0
        Top         = 480
        Width       = 3975
    End
End
Attribute VB_Name = "frmString"

```

```

Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
5 Option Explicit

Private mudtModel As Model
Private mstrVariableName As String
Private mstrStringValue As String
Private mblnOK As Boolean

10 Public Property Let Model(ByVal udtNewValue As Model)

    Set mudtModel = udtNewValue

End Property

Public Property Let VariableName(ByVal strNewValue As String)
15     mstrVariableName = strNewValue

End Property

Public Property Let StringValue(ByVal strNewValue As String)
20     mstrStringValue = strNewValue

End Property

Public Property Get StringValue() As String

    StringValue = mstrStringValue

25 End Property

Public Property Get OK() As Boolean

    OK = mblnOK

End Property

30 Private Sub Form_Load()

    mblnOK = False

```

```
frmString.Caption = "Editing string " & mstrVariableName
```

```
txtString = mstrStringValue
```

```
5   If mudtModel.IsFrozen Then  
      cmdStrOK.Enabled = False  
   End If
```

```
End Sub
```

```
10  Private Sub cmdStrOK_Click()
```

```
    mblnOK = True  
    StringValue = txtString
```

```
    Unload Me
```

```
15  End Sub
```

```
Private Sub cmdStrCancel_Click()
```

```
    Unload Me
```

```
End Sub
```

```
Private Sub txtString_GotFocus()
```

```
20  ' Automatically select all text when TextBox gets focus  
    Call txtSelectAll(txtString)
```

```
End Sub
```

```

' TCA.FRM
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Object = "{BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.1#0"; "TABCTL32.OCX"
5 Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
Begin VB.Form frmTCA
    Caption      = "ETS Test Creation Assistant"
    ClientHeight = 8310
    ClientLeft   = 165
10    ClientTop    = 735
    ClientWidth  = 11400
    LinkTopic    = "Form1"
    LockControls = -1 'True
    ScaleHeight  = 8310
15    ScaleWidth   = 11400
    StartUpPosition = 3 'Windows Default
Begin VB.Frame frmDummy
    Caption      = "Common dialog anchor"
    Height       = 855
20    Left        = 2640
    TabIndex     = 3
    Top          = 2280
    Visible      = 0 'False
    Width        = 2055
25    Begin MSComDlg.CommonDialog cdlCD
        Left      = 120
        Top       = 240
        _ExtentX  = 847
        _ExtentY  = 847
30        _Version = 393216
    End
End
Begin VB.Frame fraWord
    Height      = 8535
35    Left       = 120
    TabIndex    = 1
    Top         = 0
    Width       = 6255
End
40    Begin TabDlg.SSTab sstMainTab
        Height      = 8535
        Left        = 6480
        TabIndex     = 0
        Top         = 0

```

```

Width      = 5655
_ExtentX   = 9975
_ExtentY   = 15055
_Version    = 393216
TabHeight  = 520
BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
  Name      = "MS Sans Serif"
  Size      = 8.25
  Charset   = 0
  Weight    = 400
  Underline  = 0 'False
  Italic     = 0 'False
  Strikethrough = 0 'False
EndProperty
TabCaption(0) = "Family Overview"
TabPicture(0) = "TCA.frx":0000
Tab(0).ControlEnabled= -1 'True
Tab(0).Control(0)= "lblFamily"
Tab(0).Control(0).Enabled= 0 'False
Tab(0).Control(1)= "imlI"
Tab(0).Control(1).Enabled= 0 'False
Tab(0).Control(2)= "lblDummy"
Tab(0).Control(2).Enabled= 0 'False
Tab(0).Control(3)= "lblAccepted"
Tab(0).Control(3).Enabled= 0 'False
Tab(0).Control(4)= "lstAccepted"
Tab(0).Control(4).Enabled= 0 'False
Tab(0).Control(5)= "txtVariablize"
Tab(0).Control(5).Enabled= 0 'False
Tab(0).Control(6)= "treModels"
Tab(0).Control(6).Enabled= 0 'False
Tab(0).Control(7)= "cmdSetAttributes"
Tab(0).Control(7).Enabled= 0 'False
Tab(0).Control(8)= "lstDummy"
Tab(0).Control(8).Enabled= 0 'False
Tab(0).Control(9)= "cmdDone"
Tab(0).Control(9).Enabled= 0 'False
Tab(0).Control(10)= "cmdPrintBatch"
Tab(0).Control(10).Enabled= 0 'False
Tab(0).Control(11)= "cmdTreeExtend"
Tab(0).Control(11).Enabled= 0 'False
Tab(0).Control(12)= "cmdTreeRemove"
Tab(0).Control(12).Enabled= 0 'False
Tab(0).Control(13)= "cmdAcceptedPaste"
Tab(0).Control(13).Enabled= 0 'False

```



```

Tab(0).Control(14)= "cmdAcceptedCopy"
Tab(0).Control(14).Enabled= 0 'False
Tab(0).Control(15)= "cmdAcceptedEdit"
Tab(0).Control(15).Enabled= 0 'False
5 Tab(0).ControlCount= 16
TabCaption(1) = "Model Workshop"
TabPicture(1) = "TCA.frx":001C
Tab(1).ControlEnabled= 0 'False
Tab(1).Control(0)= "lblVariables"
10 Tab(1).Control(1)= "lblCloningConstraints"
Tab(1).Control(2)= "lblDistractor"
Tab(1).Control(3)= "cmdExportConstraints"
Tab(1).Control(4)= "cmdImportConstraints"
Tab(1).Control(5)= "cmdSaveModel"
15 Tab(1).Control(6)= "cmdTestAll"
Tab(1).Control(7)= "lstConstraints(1)"
Tab(1).Control(8)= "cmdVariableAdd"
Tab(1).Control(9)= "cmdVariableEdit"
Tab(1).Control(10)= "cmdVariableRemove"
20 Tab(1).Control(11)= "cmdVariableTest"
Tab(1).Control(12)= "cmdConstraintAdd(0)"
Tab(1).Control(13)= "cmdConstraintEdit(0)"
Tab(1).Control(14)= "cmdConstraintRemove(0)"
Tab(1).Control(15)= "cmdConstraintTest(0)"
25 Tab(1).Control(16)= "cmdConstraintAdd(1)"
Tab(1).Control(17)= "cmdConstraintEdit(1)"
Tab(1).Control(18)= "cmdConstraintRemove(1)"
Tab(1).Control(19)= "cmdConstraintTest(1)"
Tab(1).Control(20)= "cmdPrintConstraints"
30 Tab(1).Control(21)= "lstConstraints(0)"
Tab(1).Control(22)= "lstVariables"
Tab(1).Control(23)= "cmdComments"
Tab(1).ControlCount= 24
TabCaption(2) = "Generate Variants"
35 TabPicture(2) = "TCA.frx":0038
Tab(2).ControlEnabled= 0 'False
Tab(2).Control(0)= "cmdDispMakeModel"
Tab(2).Control(1)= "cmdDispDiscard"
Tab(2).Control(2)= "cmdDispDefer"
40 Tab(2).Control(3)= "cmdDispAccept"
Tab(2).Control(4)= "sldDifference"
Tab(2).Control(5)= "lstDisposition"
Tab(2).Control(6)= "cmdPrintVariants"
Tab(2).Control(7)= "cmdDisplayModel"
45 Tab(2).Control(8)= "txtNum2Generate"

```

```

Tab(2).Control(9)= "cmdGenerate"
Tab(2).Control(10)= "lblDiff"
Tab(2).Control(11)= "Label1"
Tab(2).Control(12)= "lblMed"
5 Tab(2).Control(13)= "lblLow"
Tab(2).Control(14)= "lblVariants"
Tab(2).Control(15)= "LblNumVariants"
Tab(2).ControlCount= 16
Begin VB.CommandButton cmdComments
10 Caption = "Comments"
Height = 495
Left = -70680
TabIndex = 58
ToolTipText = "Click here to print all variables and constraints."
15 Top = 3720
Width = 1215
End
Begin VB.ListBox lstVariables
20 DragIcon = "TCA.frx":0054
Height = 1635
ItemData = "TCA.frx":035E
Left = -74760
List = "TCA.frx":0360
25 Style = 1 'Checkbox
TabIndex = 57
ToolTipText = "Left button click to select a constraint. Then right button click for
constraint options."
Top = 720
Width = 3855
30 End
Begin VB.ListBox lstConstraints
DragIcon = "TCA.frx":0362
Height = 1635
Index = 0
35 ItemData = "TCA.frx":066C
Left = -74760
List = "TCA.frx":066E
Style = 1 'Checkbox
TabIndex = 56
40 ToolTipText = "Left button click to select a constraint. Then right button click for
constraint options."
Top = 3120
Width = 3855
End
45 Begin VB.CommandButton cmdAcceptedEdit

```

```

Caption      = "Edit Profile"
Height       = 255
Left         = 240
TabIndex     = 54
5  ToolTipText = "Click here to edit the profile of the selected variant."
Top          = 7300
Width        = 1335
End
Begin VB.CommandButton cmdAcceptedCopy
10  Caption      = "Copy Profile"
    Height       = 255
    Left         = 1560
    TabIndex     = 53
    ToolTipText  = "Click here to copy the profile of the selected variant."
15  Top          = 7300
    Width        = 1335
End
Begin VB.CommandButton cmdAcceptedPaste
20  Caption      = "Paste Profile"
    Height       = 255
    Left         = 2880
    TabIndex     = 52
    ToolTipText  = "Click here to paste a profile onto the currently selected variants."
25  Top          = 7300
    Width        = 1215
End
Begin VB.CommandButton cmdPrintConstraints
30  Caption      = "Print Constraints"
    Height       = 495
    Left         = -70680
    TabIndex     = 51
    ToolTipText  = "Click here to print all variables and constraints."
    Top          = 3120
    Width        = 1215
35  End
Begin VB.CommandButton cmdDispMakeModel
    Caption      = "Create Mdl."
    Height       = 255
    Left         = -71880
40  TabIndex     = 50
    ToolTipText  = "Click here to create new children of the active model using the
currently selected variants."
    Top          = 6120
    Width        = 975
45  End

```

```

Begin VB.CommandButton cmdDispDiscard
    Caption      = "Discard"
    Height       = 255
    Left         = -72840
5    TabIndex    = 49
    ToolTipText  = "Click here to discard the currently selected variants."
    Top          = 6120
    Width        = 975
End
10 Begin VB.CommandButton cmdDispDefer
    Caption      = "Defer"
    Height       = 255
    Left         = -73800
    TabIndex    = 48
15    ToolTipText  = "Click here to defer the currently selected variants."
    Top          = 6120
    Width        = 975
End
Begin VB.CommandButton cmdDispAccept
20    Caption      = "Accept"
    Height       = 255
    Left         = -74760
    TabIndex    = 47
25    ToolTipText  = "Click here to accept the currently selected variants."
    Top          = 6120
    Width        = 975
End
Begin VB.CommandButton cmdTreeRemove
30    Caption      = "Remove"
    Height       = 255
    Left         = 2160
    TabIndex    = 46
    ToolTipText  = "Click here to remove a model."
35    Top          = 3720
    Width        = 1935
End
Begin VB.CommandButton cmdTreeExtend
    Caption      = "Extend"
    Height       = 255
40    Left         = 240
    TabIndex    = 45
    ToolTipText  = "Click here to create a new child of the selected model."
    Top          = 3720
    Width        = 1935
45 End

```

```

Begin VB.CommandButton cmdConstraintTest
    Caption      = "Test"
    Height       = 255
    Index        = 1
    Left         = -71880
    TabIndex     = 44
    ToolTipText  = "Click here to test all enabled variables and distractor constraints."
    Top          = 7200
    Width        = 975
End
Begin VB.CommandButton cmdConstraintRemove
    Caption      = "Remove"
    Height       = 255
    Index        = 1
    Left         = -72840
    TabIndex     = 43
    ToolTipText  = "Click here to remove a distractor constraint."
    Top          = 7200
    Width        = 975
End
Begin VB.CommandButton cmdConstraintEdit
    Caption      = "Edit"
    Height       = 255
    Index        = 1
    Left         = -73800
    TabIndex     = 42
    ToolTipText  = "Click here to edit the currently selected distractor constraint."
    Top          = 7200
    Width        = 975
End
Begin VB.CommandButton cmdConstraintAdd
    Caption      = "Add"
    Height       = 255
    Index        = 1
    Left         = -74760
    TabIndex     = 41
    ToolTipText  = "Click here to add a distractor constraint."
    Top          = 7200
    Width        = 975
End
Begin VB.CommandButton cmdConstraintTest
    Caption      = "Test"
    Height       = 255
    Index        = 0
    Left         = -71880

```

```

    TabIndex      = 40
    ToolTipText   = "Click here to test all enabled variables and variation constraints."
    Top           = 4800
    Width         = 975
5 End
Begin VB.CommandButton cmdConstraintRemove
    Caption       = "Remove"
    Height        = 255
    Index         = 0
10    Left        = -72840
    TabIndex      = 39
    ToolTipText   = "Click here to remove the currently selected variation constraint."
    Top           = 4800
    Width         = 975
15 End
Begin VB.CommandButton cmdConstraintEdit
    Caption       = "Edit"
    Height        = 255
    Index         = 0
20    Left        = -73800
    TabIndex      = 38
    ToolTipText   = "Click here to edit the currently selected variation constraint."
    Top           = 4800
    Width         = 975
25 End
Begin VB.CommandButton cmdConstraintAdd
    Caption       = "Add"
    Height        = 255
    Index         = 0
30    Left        = -74760
    TabIndex      = 37
    ToolTipText   = "Click here to add a variation constraint."
    Top           = 4800
    Width         = 975
35 End
Begin VB.CommandButton cmdVariableTest
    Caption       = "Test"
    Height        = 255
    Left         = -71880
40    TabIndex      = 36
    ToolTipText   = "Click here to test all enabled variables."
    Top           = 2400
    Width         = 975
End
45 Begin VB.CommandButton cmdVariableRemove

```

```

Caption      = "Remove"
Height       = 255
Left         = -72840
TabIndex     = 35
5  ToolTipText = "Click here to remove the currently selected variable."
Top          = 2400
Width        = 975
End
Begin VB.CommandButton cmdVariableEdit
10  Caption      = "Edit"
    Height       = 255
    Left         = -73800
    TabIndex     = 34
    ToolTipText  = "Click here to edit the currently selected variable."
15  Top          = 2400
    Width        = 975
End
Begin VB.CommandButton cmdVariableAdd
    Caption      = "Add"
20  Height       = 255
    Left         = -74760
    TabIndex     = 33
    ToolTipText  = "Click here to add a variable."
    Top          = 2400
25  Width        = 975
End
Begin VB.CommandButton cmdPrintBatch
    Caption      = "Print All"
30  Height       = 495
    Left         = 4320
    TabIndex     = 31
    ToolTipText  = "Click here to print all variants."
    Top          = 4200
    Width        = 1215
35  End
Begin VB.CommandButton cmdDone
    Caption      = "Done"
    Height       = 495
    Left         = 4320
40  TabIndex     = 29
    ToolTipText  = "Click here when you are done with this family and are ready to send it
back to TCS."
    Top          = 1320
    Width        = 1215
45  End

```

Begin ComctlLib.Slider sldDifference

Height = 255

Left = -73440

TabIndex = 24

ToolTipText = "Select the degree of randomization desired."

Top = 1140

Width = 1935

_ExtentX = 3413

_ExtentY = 450

_Version = 327682

Max = 2

SelStart = 2

Value = 2

End

Begin VB.ListBox lstDisposition

Height = 3570

ItemData = "TCA.frx":0670

Left = -74760

List = "TCA.frx":0672

MultiSelect = 2 'Extended

TabIndex = 21

ToolTipText = "Left button click to select a variant. Then right button click for variant options."

Top = 2520

Width = 3855

End

Begin VB.CommandButton cmdPrintVariants

Caption = "Print All"

Height = 495

Left = -70680

TabIndex = 20

ToolTipText = "Click here to print all variants."

Top = 2400

Width = 1215

End

Begin VB.CommandButton cmdDisplayModel

Caption = "Display Model"

Height = 495

Left = -70680

TabIndex = 19

ToolTipText = "Click here to view the active model."

Top = 1320

Width = 1215

End

Begin VB.ListBox lstDummy


```

Height      = 255
ItemData    = "TCA.frx":0674
Left        = 4680
List        = "TCA.frx":0676
Sorted      = -1 'True
TabIndex    = 18
Top         = 7800
Visible     = 0 'False
Width       = 615

```

```
End
```

```
Begin VB.TextBox txtNum2Generate
```

```

Height      = 315
Left        = -74760
TabIndex    = 16
ToolTipText = "Enter the number variants to generate here."
Top         = 1140
Width       = 855

```

```
End
```

```
Begin VB.CommandButton cmdSetAttributes
```

```

Caption      = "Set Attributes"
Enabled      = 0 'False
Height       = 495
Left         = 4320
TabIndex     = 15
ToolTipText  = "Click here to reset the attributes for this model family."
Top          = 720
Width        = 1215

```

```
End
```

```
Begin ComctlLib.TreeView treModels
```

```

DragIcon     = "TCA.frx":0678
Height       = 2955
Left         = 240
TabIndex     = 13
ToolTipText  = "Left button click on a model to select it. Then right button click for
options."
Top          = 780
Width        = 3855
_ExtentX     = 6800
_ExtentY     = 5212
_Version     = 327682
LabelEdit    = 1
LineStyle    = 1
Style        = 7
Appearance   = 1

```

```
End
```

```

Begin VB.ListBox lstConstraints
    DragIcon      = "TCA.frx":07C2
    Height        = 1635
    Index         = 1
5    ItemData     = "TCA.frx":0ACC
    Left          = -74760
    List          = "TCA.frx":0ACE
    Style         = 1 'Checkbox
    TabIndex      = 10
10    ToolTipText = "Left button click to select a constraint. Then right button click for
constraint options."
    Top           = 5520
    Width         = 3855
End
15 Begin VB.CommandButton cmdTestAll
    Caption       = "Test All"
    Height        = 495
    Left          = -70680
    TabIndex      = 8
20    ToolTipText = "Click here to test all checked variables and constraints."
    Top           = 1320
    Width         = 1215
End
25 Begin VB.CommandButton cmdSaveModel
    Caption       = "Save Model"
    Height        = 495
    Left          = -70680
    TabIndex      = 7
30    ToolTipText = "Click here to save this model."
    Top           = 720
    Width         = 1215
End
35 Begin VB.CommandButton cmdImportConstraints
    Caption       = "Import Constraints"
    Height        = 495
    Left          = -70680
    TabIndex      = 6
40    ToolTipText = "Click here to import a variable/constraint set."
    Top           = 1920
    Width         = 1215
End
45 Begin VB.CommandButton cmdExportConstraints
    Caption       = "Export Constraints"
    Height        = 495
    Left          = -70680

```

```

        TabIndex      = 5
        ToolTipText   = "Click here to export a variable/constraint set."
        Top           = 2520
        Width         = 1215
5      End
      Begin VB.CommandButton cmdGenerate
        Caption       = "Generate"
        Height        = 495
        Left          = -70680
10     TabIndex      = 4
        ToolTipText   = "Click here to generate variants."
        Top           = 720
        Width         = 1215
      End
15     Begin VB.TextBox txtVariablize
        BackColor      = &H8000000C&
        Height         = 375
        Left           = 5880
        TabIndex       = 2
20     Text           = "Rob"
        Top            = 4740
        Visible        = 0 'False
        Width          = 615
      End
25     Begin VB.ListBox lstAccepted
        Height         = 2985
        ItemData       = "TCA.frx":0AD0
        Left           = 240
        List           = "TCA.frx":0AD2
30     MultiSelect    = 2 'Extended
        TabIndex       = 55
        ToolTipText    = "Left button click on a variant to view it. Then right button click for
options."
        Top            = 4320
35     Width          = 3855
      End
      Begin VB.Label lblAccepted
        Caption       = "Accepted variants"
        Height        = 255
40     Left          = 240
        TabIndex      = 32
        Top           = 4080
        Width         = 2535
      End
45     Begin VB.Label lblDiff

```

Caption = "Prolog randomization:"
Height = 255
Left = -73440
TabIndex = 28
Top = 840
Width = 1935

End

Begin VB.Label Label1

Caption = "High"
Height = 255
Left = -71760
TabIndex = 27
Top = 1440
Width = 495

End

Begin VB.Label lblMed

Caption = "Medium"
Height = 255
Left = -72720
TabIndex = 26
Top = 1440
Width = 735

End

Begin VB.Label lblLow

Caption = "Low"
Height = 255
Left = -73440
TabIndex = 25
Top = 1440
Width = 495

End

Begin VB.Label lblDummy

BorderStyle = 1 'Fixed Single
Height = 375
Left = 4680
TabIndex = 23
Top = 6840
Visible = 0 'False
Width = 615

End

Begin VB.Label lblVariants

Caption = "Variants"
Height = 255
Left = -74760
TabIndex = 22

```

Top      = 2280
Width    = 2055
End
Begin ComctlLib.ImageList iml1
5   Left      = 4680
    Top      = 7200
    _ExtentX = 1005
    _ExtentY = 1005
    BackColor = -2147483643
10  ImageWidth = 16
    ImageHeight = 16
    MaskColor = 12632256
    _Version = 327682
BeginProperty Images {0713E8C2-850A-101B-AFC0-4210102A8DA7}
15  NumListImages = 2
    BeginProperty ListImage1 {0713E8C3-850A-101B-AFC0-4210102A8DA7}
        Picture = "TCA.frx":0AD4
        Key = ""
    EndProperty
20  BeginProperty ListImage2 {0713E8C3-850A-101B-AFC0-4210102A8DA7}
        Picture = "TCA.frx":1026
        Key = ""
    EndProperty
EndProperty
End
25  Begin VB.Label lblNumVariants
    Caption = "Number:"
    Height = 255
    Left = -74760
30  TabIndex = 17
    Top = 900
    Width = 735
End
Begin VB.Label lblFamily
35  Caption = "Family members"
    Height = 255
    Left = 240
    TabIndex = 14
    Top = 540
40  Width = 3615
End
Begin VB.Label lblDistractor
Caption = "Distractor Constraints"
Height = 255
45  Left = -74760

```

```
    TabIndex    = 12
    Top         = 5280
    Width       = 2535
```

```
End
```

```
Begin VB.Label lblCloningConstraints
```

```
    Caption     = "Variation Constraints"
    DragIcon    = "TCA.frx":1578
    Height      = 255
    Left        = -74760
    TabIndex    = 11
    Top         = 2880
    Width       = 2535
```

```
End
```

```
Begin VB.Label lblVariables
```

```
    Caption     = "Variables"
    Height      = 255
    Left        = -74760
    TabIndex    = 9
    Top         = 480
    Width       = 855
```

```
End
```

```
End
```

```
Begin ComctlLib.StatusBar stbS
```

```
    Align       = 2 'Align Bottom
    Height      = 300
    Left        = 0
    TabIndex    = 30
    Top         = 8010
    Width       = 11400
    _ExtentX    = 20108
    _ExtentY    = 529
    SimpleText   = ""
    _Version    = 327682
```

```
BeginProperty Panels {0713E89E-850A-101B-AFC0-4210102A8DA7}
```

```
    NumPanels   = 11
```

```
BeginProperty Panel1 {0713E89F-850A-101B-AFC0-4210102A8DA7}
```

```
    Alignment   = 2
    AutoSize    = 2
    Bevel       = 0
    Object.Width      = 2117
    MinWidth        = 2117
    Text            = "Program:"
    TextSave        = "Program:"
    Key             = ""
    Object.Tag      = ""
```

```

EndProperty
BeginProperty Panel2 {0713E89F-850A-101B-AFC0-4210102A8DA7}
  Alignment    = 1
  AutoSize     = 2
  Object.Width = 1058
  MinWidth     = 1058
  Key          = ""
  Object.Tag    = ""
EndProperty
BeginProperty Panel3 {0713E89F-850A-101B-AFC0-4210102A8DA7}
  Alignment    = 2
  AutoSize     = 2
  Bevel        = 0
  Object.Width = 1773
  MinWidth     = 1764
  Text         = "Family:"
  TextSave     = "Family:"
  Key          = ""
  Object.Tag    = ""
EndProperty
BeginProperty Panel4 {0713E89F-850A-101B-AFC0-4210102A8DA7}
  Alignment    = 1
  AutoSize     = 2
  Object.Width = 2646
  MinWidth     = 2646
  Key          = ""
  Object.Tag    = ""
EndProperty
BeginProperty Panel5 {0713E89F-850A-101B-AFC0-4210102A8DA7}
  Alignment    = 2
  AutoSize     = 2
  Bevel        = 0
  Object.Width = 2117
  MinWidth     = 2117
  Text         = "Attributes:"
  TextSave     = "Attributes:"
  Key          = ""
  Object.Tag    = ""
EndProperty
BeginProperty Panel6 {0713E89F-850A-101B-AFC0-4210102A8DA7}
  Alignment    = 1
  AutoSize     = 2
  Object.Width = 1058
  MinWidth     = 1058
  Key          = ""

```

```

    Object.Tag      = ""
EndProperty
BeginProperty Panel7 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment      = 1
    AutoSize       = 2
    Object.Width    = 1058
    MinWidth       = 1058
    Key            = ""
    Object.Tag      = ""
EndProperty
BeginProperty Panel8 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    AutoSize       = 2
    Object.Width    = 1058
    MinWidth       = 1058
    Key            = ""
    Object.Tag      = ""
EndProperty
BeginProperty Panel9 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment      = 2
    AutoSize       = 2
    Bevel          = 0
    Object.Width    = 2487
    MinWidth       = 2469
    Text           = "Active Model:"
    TextSave       = "Active Model:"
    Key            = ""
    Object.Tag      = ""
EndProperty
BeginProperty Panel10 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment      = 1
    AutoSize       = 2
    Object.Width    = 450
    MinWidth       = 441
    Key            = ""
    Object.Tag      = ""
EndProperty
BeginProperty Panel11 {0713E89F-850A-101B-AFC0-4210102A8DA7}
    Alignment      = 1
    AutoSize       = 2
    Object.Width    = 2646
    MinWidth       = 2646
    Key            = ""
    Object.Tag      = ""
EndProperty
EndProperty

```



```

End
Begin VB.Menu mnuFile
    Caption      = "File"
    Begin VB.Menu mnuFileNew
        Caption    = "New"
    End
    Begin VB.Menu mnuFileOpen
        Caption    = "Open"
    End
    Begin VB.Menu mnuFileImportItem
        Caption    = "Import Locked Item"
    End
    Begin VB.Menu mnuFileSaveAs
        Caption    = "Save As"
        Visible    = 0 'False
    End
    Begin VB.Menu mnuFileSave
        Caption    = "Save"
        Visible    = 0 'False
    End
    Begin VB.Menu mnuFilePrintSetup
        Caption    = "Print Setup"
    End
    Begin VB.Menu mnuFileExit
        Caption    = "Exit"
    End
End
Begin VB.Menu mnuHelp
    Caption      = "Help"
    NegotiatePosition= 3 'Right
    Begin VB.Menu mnuHelpAbout
        Caption    = "About"
    End
End
Begin VB.Menu mnuVariables
    Caption      = "Variables"
    Visible      = 0 'False
    Begin VB.Menu mnuVariablesAdd
        Caption    = "Add"
    End
    Begin VB.Menu mnuVariablesEdit
        Caption    = "Edit"
    End
    Begin VB.Menu mnuVariablesRemove
        Caption    = "Remove"
    End
End

```

```

End
Begin VB.Menu mnuVariablesRemoveAll
    Caption    = "Remove All"
End
5  Begin VB.Menu mnuVariablesEnableAll
    Caption    = "Enable All"
End
Begin VB.Menu mnuVariablesDisableAll
    Caption    = "Disable All"
10 End
Begin VB.Menu mnuVariablesTest
    Caption    = "Test"
End
End
15 Begin VB.Menu mnuConstraints
    Caption    = "Constraints"
    Visible    = 0 'False
    Begin VB.Menu mnuConstraintsAdd
        Caption    = "Add"
20 End
    Begin VB.Menu mnuConstraintsEdit
        Caption    = "Edit"
    End
    Begin VB.Menu mnuConstraintsRemove
        Caption    = "Remove"
25 End
    Begin VB.Menu mnuConstraintsRemoveAll
        Caption    = "Remove All"
    End
    Begin VB.Menu mnuConstraintsEnableAll
        Caption    = "Enable All"
30 End
    Begin VB.Menu mnuConstraintsDisableAll
        Caption    = "Disable All"
35 End
    Begin VB.Menu mnuConstraintsTest
        Caption    = "Test"
    End
End
40 Begin VB.Menu mnuDisp
    Caption    = "Disposition"
    Visible    = 0 'False
    Begin VB.Menu mnuDispAccept
        Caption    = "Accept"
45 End

```

```

Begin VB.Menu mnuDispDefer
    Caption      = "Defer"
End
Begin VB.Menu mnuDispDiscard
    Caption      = "Discard"
End
Begin VB.Menu mnuDispMakeModel
    Caption      = "Create Model"
End
End
Begin VB.Menu mnuTree
    Caption      = "Tree"
    Visible      = 0 'False
Begin VB.Menu mnuTreeExtend
    Caption      = "Extend"
    Enabled      = 0 'False
End
Begin VB.Menu mnuTreeRemove
    Caption      = "Remove"
End
End
Begin VB.Menu mnuAccepted
    Caption      = "Accepted"
    Visible      = 0 'False
Begin VB.Menu mnuAcceptedProfile
    Caption      = "Edit profile"
End
Begin VB.Menu mnuAcceptedCopy
    Caption      = "Copy profile"
    Enabled      = 0 'False
End
Begin VB.Menu mnuAcceptedPaste
    Caption      = "Paste profile"
    Enabled      = 0 'False
End
End
End
Attribute VB_Name = "frmTCA"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

' contains family

```

Private mudtFam As Family

' word

Private mudtWord As MSWord

' prolog activex

5 Private mudtProlog As Prolog

' needed for SetAllCheckboxes sub

Private mlstCurrentListBox As ListBox

' needed so frmConstraint know which kind of constraint to create

Private mintConstrLBInd As Integer

10 ' used as a flag when mnuFileImportLockedItem calls mnuFileNew

Private mudtItemType As ItemType

' holding area for copy / paste of variant profiles

Private mudtClone As Clone

' turn full window drag back on if this is TRUE

15 Private mblnRestoreFullWindowDrag As Boolean

Public Enum EditFlags

aeNothing = 0

aeAdd = 1

aeEdit = 2

20 End Enum

Public Enum TestType

tcTestVariables = 0

tcTestVariationConstraints = 1

tcTestDistractorConstraints = 2

25 tcTestAll = 4

End Enum

' for importing/exporting variables and constraints

Private Enum ConstraintRecordLayout

crVariableIndex = 1 ' 4 byte long

30 crConstraintIndex = 5 ' 4 byte long

crVariables = 9 ' binary - variable size

End Enum

Private Enum IconImage

imSnowflake = 1

```
    imSun = 2
End Enum
```

```
' used to update status bar
```

```
Private Enum PanelIndex
```

```
    pnProgramCaption = 1
```

```
    pnProgramName = 2
```

```
    pnFamilyCaption = 3
```

```
    pnFamilyName = 4
```

```
    pnAttributesCaption = 5
```

```
    pnItemType = 6
```

```
    pnGeneric = 7
```

```
    pnProximity = 8
```

```
    pnActiveModelCaption = 9
```

```
    pnActiveModelIcon = 10
```

```
    pnActiveModelName = 11
```

```
End Enum
```

```
Public Property Get Family() As Family
```

```
    Set Family = mudtFam
```

```
End Property
```

```
Public Property Let Family(ByVal udtNewValue As Family)
```

```
    mudtFam = udtNewValue
```

```
End Property
```

```
Private Sub cmdCancel_Click()
```

```
End Sub
```

```
Private Sub cmdAcceptedCopy_Click()
```

```
    Call mnuAcceptedCopy_Click
```

```
End Sub
```

```
Private Sub cmdAcceptedEdit_Click()
```

```
    Call mnuAcceptedProfile_Click
```

```
End Sub
```

```
Private Sub cmdAcceptedPaste_Click()
```

```
    Call mnuAcceptedPaste_Click
```

```
End Sub
```

```
Private Sub cmdComments_Click()
```

```
5    frmComments.Comment = mudtFam.ActiveModel.Comments
    frmComments.Show vbModal
    mudtFam.ActiveModel.Comments = frmComments.Comment
```

```
    UpdateTab1ControlStates
```

```
10
```

```
End Sub
```

```
Private Sub cmdConstraintAdd_Click(index As Integer)
```

```
    mintConstrLBInd = index
    Call mnuConstraintsAdd_Click
```

```
15
```

```
End Sub
```

```
Private Sub cmdConstraintEdit_Click(index As Integer)
```

```
    mintConstrLBInd = index
    Call mnuConstraintsEdit_Click
```

```
End Sub
```

```
20
```

```
Private Sub cmdConstraintRemove_Click(index As Integer)
```

```
    mintConstrLBInd = index
    Call mnuConstraintsRemove_Click
```

```
End Sub
```

```
Private Sub cmdConstraintTest_Click(index As Integer)
```

```
25
```

```
    mintConstrLBInd = index
    Call mnuConstraintsTest_Click
```

```
End Sub
```

```
Private Sub cmdDispAccept_Click()
```

Call mnuDispAccept_Click

End Sub

Private Sub cmdDispDefer_Click()

Call mnuDispDefer_Click

5 End Sub

Private Sub cmdDispDiscard_Click()

Call mnuDispDiscard_Click

End Sub

Private Sub cmdDisplayModel_Click()

10 Call mudtFam.ActiveModel.OpenDoc(mudtWord)

End Sub

Private Sub cmdDispMakeModel_Click()

Call mnuDispMakeModel_Click

End Sub

15 Private Sub cmdDone_Click()

Dim intI As Integer

Dim udtClone As Clone

Dim dMode As String

Dim iType As String

20 Dim key As String

Dim Program As String

Dim root As String

Dim udtFamIni As New IniFile

Dim udtProgress As New Progress

25 If MsgBox("Prepare this family for export to TCS?", _
vbQuestion + vbYesNo) = vbNo Then

Exit Sub

End If

If mudtFam.ActiveModel Is Nothing Then

' do nothing

Else

mudtFam.ActiveModel.WriteModel

End If

' close this so it can be copied to the out directory

mudtFam.ActiveModel.CloseDoc

Call udtProgress.Init(mudtFam.Clones.Count + 2, "Preparing family for exporting to TCS...")

udtProgress.Advance

root = ExtractFileNameNoExt(mudtFam.FileName)

udtFamIni.FN = OUT_DIRECTORY & root & ".ini"

Select Case mudtFam.Program

Case prGRE

Program = "GRE"

Case prGMAT

Program = "GMAT"

Case prSAT

Program = "SAT"

Case prMR

Program = "MR"

End Select

Dim udtInIni As New IniFile

udtInIni.FN = left(mudtFam.FileName, Len(mudtFam.FileName) - 3) & _
"ini"

Dim strModelNo As String

' started with a locked item (during this session)

strModelNo = udtInIni.GetProfileString("LockedItemData", _
"TCAModelNo")

' started with an existing family (during this session)

If strModelNo = "Not Found" Then

strModelNo = udtInIni.GetProfileString("Family", _
"TCAModelNo")

End If


```
Call udtFamIni.SetKeyValuePair("TCAModelNo", strModelNo)
Call udtFamIni.SetKeyValuePair("LockedAccnum", mudtFam.AccNum)
Call udtFamIni.SetKeyValuePair("Program", Program)
```

```
Dim strProx As String
```

```
5 Select Case mudtFam.Proximity
```

```
    Case prNear
```

```
        strProx = "close"
```

```
    Case prMedium
```

```
        strProx = "medium"
```

```
10 Case prFar
```

```
    strProx = "far"
```

```
End Select
```

```
Call udtFamIni.SetKeyValuePair("Proximity", strProx)
```

```
If mudtFam.Generic Then
```

```
15 Call udtFamIni.SetKeyValuePair("Nature", "generic")
```

```
Else
```

```
    Call udtFamIni.SetKeyValuePair("Nature", "non-generic")
```

```
End If
```

```
For Each udtClone In mudtFam.Clones
```

```
20 udtClone.CloseDoc
```

```
If udtClone.IsRouted = False Then
```

```
    dMode = "TCA"
```

```
    iType = "TCA"
```

```
    Call FileCopy(IN_DIRECTORY & udtClone.FileName, _  
25 OUT_DIRECTORY & udtClone.FileName)
```

```
Else
```

```
If udtClone.DeliveryMode = dmPPT Then
```

```
    dMode = "PPT"
```

```
Else
```

```
30 dMode = "CBT"
```

```
End If
```

```
Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
```

```
Select Case mudtFam.ItemType
```

```
Case ptStandardMC
  If dMode = "PPT" Then
    iType = "MC Item"
    Call genPPT_MultChoice(udtClone, key)
  Else
    iType = "QANTDISC"
    Call genCBT_MultChoice(udtClone, key)
  End If
```

```
Case ptQuantComp
  If dMode = "PPT" Then
    iType = "QC Discrete"
    Call genPPT_QuantComp(udtClone, key)
  Else
    iType = "QANTCOMP"
    Call genCBT_QuantComp(udtClone, key)
  End If
```

```
Case ptDataSuff
  iType = "DATASUFF"
  Call genCBT_DataSuff(udtClone, key)
```

```
End Select
```

```
udtClone.CloneDoc.Close
```

```
End If
```

```
Dim udtClnIni As New IniFile
```

```
root = ExtractFileNameNoExt(udtClone.FileName)
Call udtFamIni.SetKeyValuePair("Variant", root)
```

```
udtClnIni.FN = OUT_DIRECTORY & root & ".ini"
```

```
Call udtClnIni.SetKeyValuePair("DeliveryMode", dMode)
Call udtClnIni.SetKeyValuePair("Key", udtClone.key)
Call udtClnIni.SetKeyValuePair("ItemType", iType)
Call udtClnIni.WriteProfileSection("Variant")
Call udtClnIni.WriteProfileString("Exit", " ", " ")
```

```
Set udtClnIni = Nothing
```

```
udtProgress.Advance
```

Next udtClone

' delete profiled variants from lstAccepted

With lstAccepted

intI = .ListCount - 1

Do While intI > -1

Set udtClone = mudtFam.Clones.Item(Str(.ItemData(intI)))

If udtClone.IsRouted Then

' remove the clone from the collection

Call mudtFam.Clones.Remove(Str(.ItemData(intI)))

' remove it from the list box

Call .RemoveItem(intI)

End If

intI = intI - 1

Loop

End With

mudtFam.WriteFamily

Dim fName As String

Dim strWildCard As String

For intI = 1 To treModels.Nodes.Count

root = ExtractFileNameNoExt(treModels.Nodes.Item(intI))

fName = root & ".doc"

Call udtFamIni.SetKeyValuePair("Member", fName)

Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)

fName = root & ".mdl"

Call udtFamIni.SetKeyValuePair("Member", fName)

Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)

If intI = 1 Then

fName = root & ".mdf"

strWildCard = root & ".*"

Call udtFamIni.SetKeyValuePair("Member", fName)

Call FileCopy(IN_DIRECTORY & fName, OUT_DIRECTORY & fName)

End If

Next

Call udtFamIni.WriteProfileSection("Family")

Call udtFamIni.WriteProfileString("Exit", " ", " ")

ClearControls

mudtWord.WordApp.Documents.Open FileName:=App.path & "\tcaclone.doc"
mudtWord.WordApp.Documents.Close

Kill IN_DIRECTORY & strWildCard

5

If strModelNo <> "Not Found" Then
Kill IN_DIRECTORY & strModelNo & ".*"
End If

udtProgress.Advance

10

UpdateTab0ControlStates

End Sub

Private Sub genPPT_MultChoice(udtClone As Clone, itmKey As String)

Dim docTCAModel As Document
Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

15

docTCAModel.Variables.Add "PROP_ACCNUM", "SSMCPPT"

' mudtWord.WordApp.Run ("SetAccnum")
mudtWord.WordApp.Run ("StartItem.Main")

20

Dim tabchr As String
tabchr = Chr(9)
Dim destRange As Range
Set destRange = docTCAModel.Content
destRange.find.Style = "PPTStem"
destRange.find.Execute FindText:=tabchr

' MsgBox "PPT MultChoice"

25

udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
destRange.Paste
destRange.Borders.Enable = False
destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)
destRange.Style = "PPTStem"

30

Dim respRange As Range
Dim abcde As String
abcde = "ABCDE"

Dim i As Integer

For i = 1 To 5

Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & Mid(abcde, i, 1)).Range

respRange.start = respRange.start + 4

respRange.Copy

Set destRange = docTCAModel.Content

destRange.find.Style = "PPTOptions"

destRange.find.Execute FindText:="(" & Mid(abcde, i, 1) & ")"

destRange.start = destRange.start + 4

destRange.Paste

destRange.Borders.Enable = False

destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)

destRange.Style = "PPTOptions"

Next

Dim key As String

key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text

key = Mid(key, 8, 1)

itmKey = key

For i = 1 To 5

If key = Mid(abcde, i, 1) Then

key = Format(i)

Exit For

End If

Next

Dim keyRange As Range

Dim keyStart As Long

Set keyRange = docTCAModel.Content

keyStart = keyRange.End - 1

docTCAModel.Content.InsertAfter Text:=itmKey

keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End

docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

Dim tmpFName As String

tmpFName = OUT_DIRECTORY & udtClone.FileName

docTCAModel.Variables("PROP_ACCNUM").Delete

docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"

```
docTCAModel.SaveAs tmpFName
docTCAModel.Close
```

```
End Sub
```

```
Private Sub genCBT_MultChoice(udtClone As Clone, itmKey As String)
```

```
5   Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

'   MsgBox "CBT MultChoice"

    docTCAModel.Variables.Add "PROP_ACCNUM", "SSMCCBT"
'   mudtWord.WordApp.Run ("SetAccnum")
10  mudtWord.WordApp.Run ("StartItem.Main")

    Dim tabchr As String
    tabchr = Chr(9)
    Dim destRange As Range
    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter stem here."

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    destRange.Paste
    destRange.Borders.Enable = False

    Dim respRange As Range
    Dim abcde As String
    abcde = "ABCDE"
    Dim i As Integer

    Set destRange = docTCAModel.Content
    destRange.find.Execute FindText:="Enter responses here"
25  destRange.End = destRange.End + 1
    destRange.Delete

    For i = 1 To 5

        Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & Mid(abcde, i, 1)).Range
        respRange.start = respRange.start + 4
30  respRange.Copy

        destRange.Paste
        destRange.Style = "Choice"
        destRange.InsertParagraphAfter
```

```
Set destRange = destRange.Paragraphs(1).Next.Range
```

```
Next
```

```
Dim key As String
```

```
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
```

```
key = Mid(key, 8, 1)
```

```
itmKey = key
```

```
For i = 1 To 5
```

```
    If key = Mid(abcde, i, 1) Then
```

```
        key = Format(i)
```

```
        Exit For
```

```
    End If
```

```
Next
```

```
Dim keyRange As Range
```

```
Dim keyStart As Long
```

```
Set keyRange = docTCAModel.Content
```

```
keyStart = keyRange.End - 1
```

```
docTCAModel.Content.InsertAfter Text:=itmKey
```

```
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
```

```
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange
```

```
Dim tmpFName As String
```

```
tmpFName = OUT_DIRECTORY & udtClone.FileName
```

```
docTCAModel.Variables("PROP_ACCNUM").Delete
```

```
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
```

```
Call itemKey_Store(docTCAModel, udtClone.key)
```

```
docTCAModel.SaveAs tmpFName
```

```
docTCAModel.Close
```

```
End Sub
```

```
Private Sub genPPT_QuantComp(udtClone As Clone, itmKey As String)
```

```
Dim docTCAModel As Document
```

```
Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")
```

```
' MsgBox "PPT QuantComp"
```

```
docTCAModel.Variables.Add "PROP_ACCNUM", "QCPPT"
```

```
' mudtWord.WordApp.Run ("SetAccnum")
```

```
mudtWord.WordApp.Run ("StartItem.Main")
```

```
udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy  
docTCAModel.Tables(1).Cell(Row:=1, Column:=2).Range.Paste  
docTCAModel.Tables(1).Cell(Row:=1, Column:=2).Range.Style = "PPTQC StimCentered"
```

```
5 udtClone.CloneDoc.Bookmarks("tca_ColumnA").Range.Copy  
docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Paste
```

```
udtClone.CloneDoc.Bookmarks("tca_ColumnB").Range.Copy  
docTCAModel.Tables(1).Cell(Row:=2, Column:=4).Range.Paste
```

```
10 docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Style = "PPTQC AB"  
docTCAModel.Tables(1).Cell(Row:=2, Column:=4).Range.Style = "PPTQC AB"
```

```
Dim key As String  
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text  
key = Mid(key, 8, 1)  
itmKey = key
```

```
15 Dim abcde As String  
abcde = "ABCDE"  
Dim i As Integer
```

```
For i = 1 To 5  
20 If key = Mid(abcde, i, 1) Then  
key = Format(i)  
Exit For  
End If  
Next
```

```
25 Dim keyRange As Range  
Dim keyStart As Long  
Set keyRange = docTCAModel.Content  
keyStart = keyRange.End - 1
```

```
docTCAModel.Content.InsertAfter Text:=itmKey  
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End  
30 docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange
```

```
Dim tmpFName As String  
tmpFName = OUT_DIRECTORY & udtClone.FileName
```

```
docTCAModel.Variables("PROP_ACCNUM").Delete  
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
```



```
docTCAModel.SaveAs tmpFName
docTCAModel.Close
```

```
End Sub
```

```
Private Sub genCBT_QuantComp(udtClone As Clone, itmKey As String)
```

```
5    Dim docTCAModel As Document
    Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "\TCAClone.DOC")

'   MsgBox "CBT QuantComp"

    docTCAModel.Variables.Add "PROP_ACCNUM", "QCCBT"
'   mudtWord.WordApp.Run ("SetAccnum")
10    mudtWord.WordApp.Run ("StartItem.Main")

    udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=1, Column:=1).Range.Paste

    udtClone.CloneDoc.Bookmarks("tca_ColumnA").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=2, Column:=1).Range.Paste

15    udtClone.CloneDoc.Bookmarks("tca_ColumnB").Range.Copy
    docTCAModel.Tables(1).Cell(Row:=2, Column:=2).Range.Paste

    Dim key As String
    key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
    key = Mid(key, 8, 1)
20    itmKey = key

    Dim abcde As String
    abcde = "ABCDE"
    Dim i As Integer

    For i = 1 To 5
25        If key = Mid(abcde, i, 1) Then
            key = Format(i)
            Exit For
        End If
    Next

30    Dim keyRange As Range
    Dim keyStart As Long
    Set keyRange = docTCAModel.Content
    keyStart = keyRange.End - 1
```

```

docTCAModel.Content.InsertAfter Text:=itmKey
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

```

```

Dim tmpFName As String
5 tmpFName = OUT_DIRECTORY & udtClone.FileName

```

```

docTCAModel.Variables("PROP_ACCNUM").Delete
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
Call itmKey_Store(docTCAModel, udtClone.key)
docTCAModel.SaveAs tmpFName
10 docTCAModel.Close

```

```

End Sub

```

```

Private Sub genCBT_DataSuff(udtClone As Clone, itmKey As String)

```

```

Dim docTCAModel As Document
Set docTCAModel = mudtWord.WordApp.Documents.Open(App.path & "TCAClone.DOC")

```

```

15 ' MsgBox "CBT DataSuff"

```

```

docTCAModel.Variables.Add "PROP_ACCNUM", "DSCBT"
' mudtWord.WordApp.Run ("SetAccnum")
mudtWord.WordApp.Run ("StartItem.Main")

```

```

Dim tabchr As String
20 tabchr = Chr(9)
Dim destRange As Range
Set destRange = docTCAModel.Content
destRange.find.Execute FindText:="Enter stem here."

```

```

25 udtClone.CloneDoc.Bookmarks("tca_Stem").Range.Copy
destRange.Paste

```

```

' destRange.Borders.Enable = False
' destRange.ParagraphFormat.LeftIndent = InchesToPoints(0.25)

```

```

Set destRange = docTCAModel.Content
destRange.find.Execute FindText:="Enter Data Sufficiency Statement 1 here, then press
30 return."

```

```

' udtClone.CloneDoc.Bookmarks("tca_fStatement").Range.Copy
Dim srcRange As Range
Set srcRange = udtClone.CloneDoc.Bookmarks("tca_fStatement").Range

```

```
srcRange.End = srcRange.End - 1
```

```
If Len(srcRange.Text) > 0 Then
```

```
    srcRange.Copy
```

```
    destRange.Paste
```

```
End If
```

```
destRange.Collapse Direction:=wdCollapseEnd
```

```
destRange.InsertParagraphAfter
```

```
destRange.Collapse Direction:=wdCollapseEnd
```

```
Set srcRange = udtClone.CloneDoc.Bookmarks("tca_sStatement").Range
```

```
srcRange.End = srcRange.End - 1
```

```
If Len(srcRange.Text) > 0 Then
```

```
    srcRange.Copy
```

```
    destRange.Paste
```

```
End If
```

```
Dim n As Integer
```

```
n = docTCAModel.ListParagraphs.Count
```

```
While n > 2
```

```
    Set destRange = docTCAModel.ListParagraphs(n).Range
```

```
    destRange.Delete
```

```
    n = n - 1
```

```
Wend
```

```
Dim key As String
```

```
key = udtClone.CloneDoc.Bookmarks("tca_Key").Range.Text
```

```
key = Mid(key, 8, 1)
```

```
itmKey = key
```

```
Dim abcde As String
```

```
abcde = "ABCDE"
```

```
Dim i As Integer
```

```
For i = 1 To 5
```

```
    If key = Mid(abcde, i, 1) Then
```

```
        key = Format(i)
```

```
        Exit For
```

```
    End If
```

```
Next
```

```
Dim keyRange As Range
```

```
Dim keyStart As Long
```

```
Set keyRange = docTCAModel.Content
```

```
keyStart = keyRange.End - 1
```

```

docTCAModel.Content.InsertAfter Text:=itmKey
keyRange.SetRange start:=keyStart, End:=docTCAModel.Content.End
' docTCAModel.Bookmarks.Add Name:="prop_Key", Range:=keyRange

```

```

Dim tmpFName As String
5 tmpFName = OUT_DIRECTORY & udtClone.FileName

```

```

docTCAModel.Variables("PROP_ACCNUM").Delete
docTCAModel.Variables.Add "PROP_ACCNUM", "TCAVARNT"
Call itemKey_Store(docTCAModel, udtClone.key)
docTCAModel.SaveAs tmpFName
10 docTCAModel.Close

```

```

End Sub

```

```

Private Sub itemKey_Store(doc As Document, ByVal key As String)

```

```

    Dim i As Integer

    For i = 1 To 5
15     If key = Mid("ABCDE", i, 1) Then
        key = Format(i)
        Exit For
    End If
    Next

```

```

20 doc.Variables.Add "ItemKeyStore", key

```

```

End Sub

```

```

Private Sub cmdPrintConstraints_Click()

```

```

    Dim udtV As Variable
    Dim udtC As Constraint
25 Dim udtVI As VarInteger
    Dim udtVR As VarReal
    Dim udtVF As VarFraction
    Dim udtVS As VarString
    Dim udtP As New PrintModel
30 Dim varS As Variant
    Dim varS1 As Variant
    Dim udtSS As SubString
    Dim intI As Integer

```

```

35 udtP.ModelName = ExtractFileNameNoExt(mudtFam.ActiveModel.FileName)

```

Call udtP.PrintString("Variables:", 1)

For Each udtV In mudtFam.ActiveModel.Variables

5 Call udtP.PrintString("Variable name: " & udtV.name, 2)

Select Case udtV.Typ

Case vtInteger

Call udtP.PrintString("Type: Integer", 3)

Case vtReal

10 Call udtP.PrintString("Type: Real", 3)

Case vtFraction

Call udtP.PrintString("Type: Fraction", 3)

Case vtString

Call udtP.PrintString("Type: String", 3)

15 Case vtUntyped

Call udtP.PrintString("Type: Untyped", 3)

End Select

If udtV.Enabled Then

Call udtP.PrintString("Status: Enabled", 3)

20 Else

Call udtP.PrintString("Status: Disabled", 3)

End If

If udtV.Checksum Then

Call udtP.PrintString("Checksum: Enabled", 3)

25 Else

Call udtP.PrintString("Checksum: Disabled", 3)

End If

Select Case udtV.Typ

Case vtInteger

30 Set udtVI = udtV

If udtVI.IsIndependent Then

Call udtP.PrintString("Is independent = True," & _

" Range: from " & udtVI.From & _

" to " & udtVI.To & _

35 " by " & udtVI.By, 3)

Else

Call udtP.PrintString("Is independent = False", 3)

End If

Case vtReal

40 Set udtVR = udtV

If udtVR.IsIndependent Then

Call udtP.PrintString("Is independent = True," & _

" Range: from " & udtVR.From & _

" to " & udtVR.To & _

45 " by " & udtVR.By, 3)

```

Else
    Call udtP.PrintString("Is independent = False", 3)
End If
If udtVR.IsOnGrid Then
    Call udtP.PrintString("Force on grid value: True", 3)
Else
    Call udtP.PrintString("Force on grid value: False", 3)
End If
Call udtP.PrintString("# Decimal places: " & _
    Str(udtVR.DecimalPlaces), 3)
If udtVR.TrailingZeros Then
    Call udtP.PrintString("Display trailing zeros: True", 3)
Else
    Call udtP.PrintString("Display trailing zeros: False", 3)
End If
Case vtFraction
    Set udtVF = udtV
    If udtVF.IsIndependent Then
        Call udtP.PrintString("Is independent = True," & _
            " Range: from " & udtVF.FromNumerator & _
            "/" & udtVF.FromDenominator & _
            " to " & udtVF.ToNumerator & _
            "/" & udtVF.ToDenominator & _
            " by " & udtVF.ByNumerator & _
            "/" & udtVF.ByDenominator, 3)
    Else
        Call udtP.PrintString("Is independent = False", 3)
    End If
    If udtVF.MixedNumbers Then
        Call udtP.PrintString("Display mixed number: True", 3)
    Else
        Call udtP.PrintString("Display mixed number: False", 3)
    End If
Case vtString
    Set udtVS = udtV
    If udtVS.IsIndexed Then
        Call udtP.PrintString("Indexed: True", 3)
        Call udtP.PrintString("Value Sets:", 3)
        For Each varS In udtVS.StringCollection
            Set udtSS = New SubString
            udtSS.Delimiter = Chr(STRING_DELIMITER)
            udtSS.StringValue = varS
            Call udtP.PrintString("Values:", 4)
            intI = 1
            For Each varS1 In udtSS.StringCollection

```

```

        Call udtP.PrintString(Str(intI) & ". " & varS1, 5)
        intI = intI + 1
    Next varS1
    Next varS
5    Else
        Call udtP.PrintString("Indexed: False", 3)
        Call udtP.PrintString("Values:", 3)
        For Each varS In udtVS.StringCollection
            Call udtP.PrintString(varS, 4)
10        Next varS
        End If
        Case vtUntyped
    End Select

15    Next udtV

    Call udtP.PrintString("Constraints:", 1)

    Call udtP.PrintString("Variation constraints:", 2)
20    For Each udtC In mudtFam.ActiveModel.Constraints
        If udtC.ConstraintType = ctVariation Then
            Call udtP.PrintString("Constraint: " & udtC.ConstraintString, 3)
            If udtC.Enabled Then
25                Call udtP.PrintString("Status: Enabled", 4)
            Else
                Call udtP.PrintString("Status: Disabled", 4)
            End If
        End If
    Next udtC
30
    ' exit if not MC
    If Not mudtFam.ItemType = ptStandardMC Then Exit Sub

35    Call udtP.PrintString("Distractor constraints:", 2)

    For Each udtC In mudtFam.ActiveModel.Constraints
        If udtC.ConstraintType = ctDistractor Then
            Call udtP.PrintString("Constraint: " & udtC.ConstraintString, 3)
40            If udtC.Enabled Then
                Call udtP.PrintString("Status: Enabled", 4)
            Else
                Call udtP.PrintString("Status: Disabled", 4)
            End If
45        End If
    End If

```

Next udtC

End Sub

Private Sub cmdSetAttributes_Click()

5 frmAttributes.Show vbModal

If frmAttributes.OK Then

 mudtFam.Generic = frmAttributes.Generic

 mudtFam.Proximity = frmAttributes.Proximity

 mudtFam.IsDirty = True

10 ' save family

 mudtFam.WriteFamily

 UpdateFamilyAttributes

End If

End Sub

15 Private Sub cmdTreeExtend_Click()

 Call mnuTreeExtend_Click

End Sub

Private Sub cmdTreeRemove_Click()

 Call mnuTreeRemove_Click

20 End Sub

Private Sub cmdVariableAdd_Click()

 Call mnuVariablesAdd_Click

 frmVariable.Model = mudtFam.ActiveModel

 frmVariable.ListBox = lstVariables

25 frmVariable.Show vbModal

 UpdateTab1ControlStates

End Sub

Private Sub cmdVariableEdit_Click()


```
Call mnuVariablesEdit_Click
frmVariable.Model = mudtFam.ActiveModel
frmVariable.ListBox = lstVariables
```

```
If lstVariables.ListIndex >= 0 Then ' Make sure list item is selected
```

```
5 ' Set the key for access by frmVariable
```

```
With lstVariables
```

```
    frmVariable.Variable = _
```

```
        mudtFam.ActiveModel.Variables.Item(Str(.ItemData(.ListIndex)))
```

```
End With
```

```
10 frmVariable.Show vbModal
```

```
End If
```

```
UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub cmdVariableRemove_Click()
```

```
15 Call mnuVariablesRemove_Click
```

```
End Sub
```

```
Private Sub cmdVariableTest_Click()
```

```
Call mnuVariablesTest_Click
```

```
End Sub
```

```
20 Private Sub Form_Initialize()
```

```
    frmSplash.Show
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    ' to trap cancels
```

```
25 cdlCD.CancelError = True
```

```
    ' Create Word Object
```

```
Set mudtWord = New MSWord
```

```
    ' get rid of the kill file if it exists, as it will prevent
```

```
30 ' StartProlog from working
```

DestroyKillFile

' Create the Prolog object

If mudtProlog Is Nothing Then

Set mudtProlog = CreateObject("AXProlog.Prolog")

If Not mudtProlog.StartProlog Then

Call MsgBox("Prolog cannot be started.", vbExclamation, "Prolog error")

End If

End If

treModels.ImageList = imlI

frmSplash.UnloadMe

Me.Show

UpdateTab0ControlStates

' ' copies ied files from a holding area, as TCS deletes them for

' ' reasons unknown.

Call Kill("c:\tcs\working*.ied")

Call FileCopy("c:\tcs\tcaied\dscbt.ied", "c:\tcs\working\dscbt.ied")

Call Shell("attrib -r c:\tcs\working\dscbt.ied", vbHide)

Call FileCopy("c:\tcs\tcaied\qccbt.ied", "c:\tcs\working\qccbt.ied")

Call FileCopy("c:\tcs\tcaied\qcppt.ied", "c:\tcs\working\qcppt.ied")

Call FileCopy("c:\tcs\tcaied\ssmccbt.ied", "c:\tcs\working\ssmccbt.ied")

Call FileCopy("c:\tcs\tcaied\ssmcppt.ied", "c:\tcs\working\ssmcppt.ied")

End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)

Call sstMainTab_MouseMove(Button, Shift, X, Y)

End Sub

Private Sub Form_Resize()

' if minimized, don't resize

If Me.WindowState = vbMinimized Then Exit Sub

Dim udtW As New Win32API

Dim result As Long

'Turn off full window drag if it's on

```

If udtW.IsFullWindowDragOn Then
    udtW.TurnOffFullWindowDrag
    mblnRestoreFullWindowDrag = True
End If

```

```

5      ' adjust horizontals
      fraWord.left = 120
      fraWord.Width = Me.Width - sstMainTab.Width - 360
      sstMainTab.left = fraWord.Width + 180

      'adjust verticals
10     fraWord.Height = Me.Height - fraWord.top - stbS.Height - 700 ' approx title bar height
      sstMainTab.Height = fraWord.Height

      mudtWord.Resize

```

End Sub

```

15 Private Sub Form_Unload(Cancel As Integer)

    ' if no active family, hit the road
    If mudtFam Is Nothing Then
        ' do nothing
    Else
20         mudtFam.WriteFamily
        If mudtFam.ActiveModel Is Nothing Then ' see if an active model has been set
            ' do nothing
        Else
25             mudtFam.ActiveModel.CloseDoc
            KillVariants ' Get rid of any variants left on tab 3
            mudtFam.ActiveModel.WriteModel ' save the active model
        End If
    End If

    ' close all docs
30     mudtWord.CloseAllDocs

    ' Kill Word before frmTCA is unloaded to prevent automation error
    Set mudtWord = Nothing

    ' force event
    Call sstMainTab_MouseMove(1, 1, 1, 1)

35     ' To cleanly shut down AXProlog on W95, 98 boxes
    mudtProlog.EndProlog

```

```
' End required by NT 4.0 to shut down TCA successfully!
End
```

```
End Sub
```

```
Private Sub lstVariables_ItemCheck(Item As Integer)
```

```
5 With lstVariables
  If lstVariables.ListCount = 0 Then Exit Sub ' this prevents an error
  If mudtFam.ActiveModel.IsFrozen Then
    .Selected(Item) = _
      mudtFam.ActiveModel.Variables.Item(Str(.ItemData(Item))).Enabled
10 Else
    mudtFam.ActiveModel.Variables.Item(Str(.ItemData(Item))).Enabled = _
      .Selected(Item)
  End If
End With
```

```
15 UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub lstVariables_MouseDown(Button As Integer, Shift As Integer, _
  X As Single, Y As Single)
```

```
Dim strIndex As String
```

```
20 Set mlstCurrentListBox = lstVariables
```

```
If Button = vbRightButton Then
  frmVariable.AddEditFlag = aeNothing
  PopupMenu mnuVariables ' Pull up popup menu for variable window
  frmVariable.Model = mudtFam.ActiveModel
25 frmVariable.ListBox = lstVariables
  Select Case frmVariable.AddEditFlag
    Case aeEdit
      If lstVariables.ListIndex >= 0 Then ' Make sure list item is selected
        ' Set the key for access by frmVariable
30 With lstVariables
          frmVariable.Variable = _
            mudtFam.ActiveModel.Variables.Item(Str(.ItemData(.ListIndex)))
        End With
        frmVariable.Show vbModal
35 End If
    Case aeAdd
```

```

        frmVariable.Show vbModal
    End Select
End If

End Sub

5 Private Sub lstConstraints_ItemCheck(index As Integer, Item As Integer)

    Dim strKey As String

    With lstConstraints(index)
        If .ListCount = 0 Then Exit Sub ' prevents error if listbox is empty
        If mudtFam.ActiveModel.IsFrozen Then
10         .Selected(Item) = _
            mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(Item))).Enabled
        Else
            mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(Item))).Enabled = _
                .Selected(Item)
15         End If
    End With

    UpdateTab1ControlStates

End Sub

' provide right button menu options
20 Private Sub lstConstraints_MouseDown(index As Integer, Button As Integer, _
    Shift As Integer, X As Single, Y As Single)

    Dim strIndex As String

    Set mlstCurrentListBox = lstConstraints(index)
    mintConstrLBInd = index

25 Call UpdateTab1ControlStates(index)

    If Button = vbRightButton Then
        PopupMenu mnuConstraints
    Else
        If mudtFam.ActiveModel.IsFrozen = False Then
30         lstConstraints(index).Drag
        End If
    End If

End Sub

End Sub

```

' Enable drag and drop between constraint list boxes

Private Sub lstConstraints_DragDrop(index As Integer, Source As Control, _
X As Single, Y As Single)

If Source.ListCount = 0 Then

Exit Sub

End If

If index <> Source.index Then ' Assure that it's another listbox!

Dim udtConstraint As Constraint

Dim strKey As String

strKey = Str(Source.ItemData(Source.ListIndex))

With lstConstraints(index)

' Add the dragged constraint to the end of the target listbox

.List(.ListCount) = Source.List(Source.ListIndex)

' Update the index in the new listbox entry

.ItemData(.ListCount - 1) = Source.ItemData(Source.ListIndex)

End With

' Find the constraint object being moved and update it's "type" in the collection

Set udtConstraint = mudtFam.ActiveModel.Constraints.Item(strKey)

udtConstraint.ConstraintType = index

' Delete the dragged constraint from the source listbox

Call Source.RemoveItem(Source.ListIndex)

End If

UpdateTab1ControlStates

End Sub

Private Sub lstDisposition_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)

Dim udtClone As Clone

If Button = vbRightButton Then

PopupMenu mnuDisp

Else

With lstDisposition

If .ListCount > 0 Then ' a valid selection has been made

```

        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(.ListIndex)))
        Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
    End If
End With
5   End If
End Sub

```

```

Private Sub lstAccepted_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

```

```

    Static udtClone As Clone

```

```

10   If Button = vbRightButton Then
    '   With lstAccepted
    '       If .SelCount = 1 Then
    '           mnuAcceptedProfile.Enabled = True
    '           mnuAcceptedCopy.Enabled = True
15   '           Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
    '           Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
    '           Set udtClone = Nothing
    '       Else
    '           mnuAcceptedProfile.Enabled = False
20   '           mnuAcceptedCopy.Enabled = False
    '       End If
    '   End With
    PopupMenu mnuAccepted
Else ' left button click
25   If udtClone Is Nothing Then
    ' do nothing
    Else
        udtClone.CloseDoc
        Set udtClone = Nothing
30   End If
    With lstAccepted
        If .ListCount > 0 Then
            Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
            Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
35   End If
    End With
End If

    UpdateTab0ControlStates

End Sub

```

```
Private Sub cmdSaveModel_Click()
```

```
    If mudtFam.ActiveModel.IsDirty Then  
        mudtFam.ActiveModel.WriteModel  
        KillVariants 'delete any variants on tab 3  
5    End If
```

```
    UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub cmdTestAll_Click()
```

```
    cmdSaveModel_Click ' force a save  
10    Call TestConstraints(tcTestAll)
```

```
End Sub
```

```
Private Sub cmdImportConstraints_Click()
```

```
    Dim strFN As String
```

```
    With cdlCD
```

```
        .FileName = ""  
        .CancelError = True  
        .DialogTitle = "Import constraints from file"  
        .Filter = "Constraint Files (*.con)|*.con|"  
        .DefaultExt = ".con"  
20        .InitDir = "c:\tcs\tca\constraints"  
        .Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly  
        On Error GoTo Cancel ' trap the Cancel button  
        .ShowOpen  
        On Error GoTo 0 ' reset the error  
25        strFN = .FileName
```

```
    End With
```

```
    ' exit if there's no file name
```

```
    If Len(strFN) = 0 Then  
        Exit Sub  
30    End If
```

```
    ' create a new collection of imported variables
```

```
    Dim udtCVariables As New CVariables
```


Call udtCVariables.ReadCollection(strFN, crVariableIndex, crConstraintIndex)

' add the imported variables to the main collection

Dim udtNewVar As Variable

For Each udtNewVar In udtCVariables

If mudtFam.ActiveModel.Variables.UniqueName(udtNewVar.name) Then

Call mudtFam.ActiveModel.Variables.AddObject(udtNewVar)

With lstVariables

' Add the new variable to the variable list box

Call .AddItem(udtNewVar.ScreenFormat)

' Set ItemData to index value of the variable object

.ItemData(.ListCount - 1) = udtNewVar.index

' Set the check box.

.Selected(.ListCount - 1) = udtNewVar.Enabled

End With

Else

Call MsgBox("Variable " & udtNewVar.name & " will not be imported.", _
vbExclamation, "Variable not unique")

End If

Next udtNewVar

' read the imported constraints into a new collection

Dim udtCConstraints As New CConstraints

Call udtCConstraints.ReadCollection(strFN, crConstraintIndex, READ_UNTIL_EOF)

' add the imported constraints

Dim udtNewCon As Constraint

For Each udtNewCon In udtCConstraints

If mudtFam.ActiveModel.Constraints.UniqueConstraint(udtNewCon.ConstraintString)

Then

Call mudtFam.ActiveModel.Constraints.AddObject(udtNewCon)

With lstConstraints(udtNewCon.ConstraintType)

' Add the new variable to the variable list box

Call .AddItem(udtNewCon.ConstraintString)

' Set ItemData to index value of the variable object

.ItemData(.ListCount - 1) = udtNewCon.index

' Check the check box

.Selected(.ListCount - 1) = udtNewCon.Enabled

```

        End With
    Else
        Call MsgBox("Constraint " & udtNewCon.ConstraintString & " will not be imported.", _
            vbExclamation, "Constraint not unique")
5    End If
Next udtNewCon

```

```

Cancel:
Exit Sub

```

```

End Sub

```

```

10 Private Sub cmdExportConstraints_Click()

```

```

    Dim strFN As String

```

```

    With cdlCD

```

```

        .FileName = ""

```

```

        .DialogTitle = "Export constraints to file"

```

```

        .Filter = "Constraint Files (*.con)|*.con|"

```

```

        .DefaultExt = ".con"

```

```

        .InitDir = "c:\tcs\tca\constraints"

```

```

        .Flags = cdlOFNOverwritePrompt + cdlOFNHideReadOnly

```

```

        On Error GoTo Cancel ' trap the Cancel button

```

```

        .ShowSave

```

```

        On Error GoTo 0 ' reset

```

```

        strFN = .FileName

```

```

    End With

```

```

    Dim lngEndPos As Long

```

```

25    If Len(strFN) > 0 Then

```

```

        lngEndPos = mudtFam.ActiveModel.Variables.WriteCollection(strFN, crVariableIndex,
crVariables)

```

```

        Call mudtFam.ActiveModel.Constraints.WriteCollection(strFN, crConstraintIndex,
lngEndPos)

```

```

30    End If

```

```

Cancel:
Exit Sub

```

```

End Sub

```

```

Private Sub cmdPrintBatch_Click()

```

```
Dim blnTF As Boolean
Dim udtClone As Clone
```

```
If mudtWord.WordApp.Documents.Count = 0 Then
    mudtWord.WordApp.Documents.Open FileName:=App.path & "\printing.doc"
5    blnTF = True
End If
```

```
For Each udtClone In mudtFam.Clones
    mudtWord.WordApp.PrintOut FileName:=IN_DIRECTORY & udtClone.FileName
Next udtClone
```

```
10    If blnTF Then
        mudtWord.WordApp.Documents.Close
    End If
```

```
End Sub
```

```
Private Sub cmdPrintVariants_Click()
```

```
15    Dim blnTF As Boolean
    Dim udtClone As Clone
```

```
If mudtWord.WordApp.Documents.Count = 0 Then
    mudtWord.WordApp.Documents.Open FileName:=App.path & "\printing.doc"
    blnTF = True
20    End If
```

```
For Each udtClone In mudtFam.ActiveModel.Clones
    mudtWord.WordApp.PrintOut FileName:=IN_DIRECTORY & udtClone.FileName
Next
```

```
If blnTF Then
25    mudtWord.WordApp.Documents.Close
End If
```

```
End Sub
```

```
Private Sub cmdGenerate_Click()
```

```
Dim udtClone As New Clone
```

```
30    Me.Enabled = False ' disable frmTCA to make next form seem modal
    frmProlog.Caption = "Generating " & txtNum2Generate & " variants"
    frmProlog.lblProlog.Caption = "Click Abort to terminate variant generation."
```

```

frmProlog.Show ' show form modeless so execution continues
Me.MousePointer = vbHourglass
Call mudtFam.ActiveModel.GenerateClones(mudtWord, mudtProlog, _
    CInt(txtNum2Generate), sldDifference)
5 Me.MousePointer = vbDefault
frmProlog.Kill ' destroy frmProlog
Me.Enabled = True

```

```

If lstDisposition.ListCount > 0 Then
    With lstDisposition
10         .Selected(.ListCount - 1) = True
        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(.ListCount - 1)))
        Call udtClone.OpenDoc(mudtWord, IN_DIRECTORY)
    End With
End If

```

```

15 UpdateTab2ControlStates

```

```

End Sub

```

```

Private Sub mnuDispAccept_Click()

```

```

    Dim udtClone As Clone
    Dim nodN As Node
20 Dim intI As Integer
    Dim strFN As String

```

```

    With lstDisposition
        If .SelCount > 0 Then ' make sure something's selected
            For intI = 0 To .ListCount - 1 ' for multiselect

```

```

25         If .Selected(intI) Then
            strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileName)

```

```

            ' confirm this operation
30         If MsgBox("Accept variant " & strFN & "?", _
            vbQuestion + vbYesNo, "Confirm") = vbNo Then
            .Selected(intI) = False
        End If
    End If

```

```

35     If .Selected(intI) Then
        ' get object from active model's clone collection
        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
        ' close the document, if it's open
        udtClone.CloseDoc
    End If
End Sub

```

```

        ' remove it from the active model's collection
        Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
        ' save the checksum in the model
        Call mudtFam.ActiveModel.AddChecksum(udtClone.Checksum)
5      ' add it to the family clone collection
        Call mudtFam.Clones.AddObj(udtClone)
        ' add it to the accepted list box
        Call lstAccepted.AddItem(ExtractFileName(udtClone.FileName))
        ' add key to itemdata
10     lstAccepted.ItemData(lstAccepted.ListCount - 1) = udtClone.index
        ' freeze the model
        mudtFam.ActiveModel.FreezeModel
        ' update the icon
        Set nodN = treModels.Nodes.Item(ModelKey(mudtFam.ActiveModel.FileName))
15     nodN.Image = imSnowflake
        stbS.Panels(pnActiveModelIcon).Picture = imlI.ListImages(nodN.Image).Picture
        Call mudtFam.ActiveModel.CloseDoc
        Call mudtFam.ActiveModel.OpenDoc(mudtWord)
    End If
20   Next intI
    For intI = .ListCount - 1 To 0 Step -1
        If .Selected(intI) Then
            ' remove the entry from the disposition list box
            Call .RemoveItem(intI)
25         End If
    Next intI
End If
End With

UpdateTab0ControlStates
30 UpdateTab1ControlStates
UpdateTab2ControlStates

End Sub

Private Sub mnuDispDefer_Click()

    Dim udtClone As Clone
35     Dim intI As Integer
    Dim strFN As String

    With lstDisposition
        If .SelCount > 0 Then ' make sure somethings selected
            For intI = 0 To .ListCount - 1 ' for multiselect
40                 If .Selected(intI) Then

```

```

        strFN =
ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa
me)

```

```

        ' confirm this operation
5      If MsgBox("Defer variant " & strFN & "?", _
        vbQuestion + vbYesNo, "Confirm") = vbNo Then
        .Selected(intI) = False
        End If
    End If
10    If .Selected(intI) Then
        ' get object from active model's clone collection
        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
        ' close the document
        udtClone.CloseDoc
15        ' delete the clone file
        Kill IN_DIRECTORY & udtClone.FileName
        ' remove the clone from the active model's collection
        Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
    End If
20    Next intI
    For intI = .ListCount - 1 To 0 Step -1 ' for multiselect
        If .Selected(intI) Then
            ' remove the entry from the disposition list box
            Call .RemoveItem(intI)
25        End If
    Next intI
End If
End With

```

```

    UpdateTab2ControlStates

```

```

30 End Sub

```

```

Private Sub mnuDispDiscard_Click()

```

```

    Dim udtClone As Clone
    Dim intI As Integer
    Dim strFN As String

```

```

35    With lstDisposition
        If .SelCount > 0 Then ' make sure somethings selected
            For intI = 0 To .ListCount - 1 ' for multiselect
                If .Selected(intI) Then
                    strFN =
40    ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(lstDisposition.ItemData(intI))).FileNa

```

me)

```
' confirm this operation
If MsgBox("Discard variant " & strFN & "?", _
vbQuestion + vbYesNo, "Confirm") = vbNo Then
    .Selected(intI) = False
End If
End If
If .Selected(intI) Then
    ' get object from active model's clone collection
    Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
    ' save the checksum in the model
    Call mudtFam.ActiveModel.AddChecksum(udtClone.Checksum)
    ' close the document
    udtClone.CloseDoc
    ' delete the clone file
    Kill IN_DIRECTORY & udtClone.FileName
    ' remove the clone from the active model's collection
    Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
End If
Next intI
For intI = .ListCount - 1 To 0 Step -1 ' for multiselect
    If .Selected(intI) Then
        ' remove the entry from the disposition list box
        Call .RemoveItem(intI)
    End If
Next intI
End If
End With

UpdateTab2ControlStates
```

End Sub

Private Sub mnuDispMakeModel_Click()

```
Dim udtClone As Clone
Dim strNewFN As String
Dim strKey As String
Dim strNewKey As String
Dim udtM As Model
Dim nodN As Node
Dim intI As Integer
Dim strFN As String
```

With lstDisposition

```

    If .SelCount > 0 Then ' make sure somethings selected
        For intI = 0 To .ListCount - 1 ' for multiselect
            If .Selected(intI) Then
                strFN =
5      ExtractFileName(mudtFam.ActiveModel.Clones.Item(Str(1stDisposition.ItemData(intI))).FileNa
me)
                ' confirm this operation
                If MsgBox("Create a new model from variant " & strFN & "?", _
                    vbQuestion + vbYesNo, "Confirm") = vbNo Then
10                 .Selected(intI) = False
                End If
            End If
        End If
        If .Selected(intI) Then
            ' get object from active model's clone collection
15         Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
            ' close the document
            udtClone.CloseDoc
            strKey = ModelKey(udtClone.FileName)
            ' find the next key for this parent model
20         strNewKey = NextModelKey(udtClone.FileName)
            ' add the child to the tree
            strNewFN = ModelEmbedKey(udtClone.FileName, strNewKey)
            Set nodN = treModels.Nodes.Add(strKey, twwChild, strNewKey, strNewFN)
            nodN.Expanded = True
25         nodN.sorted = True
            nodN.Image = imSun
            ' copy the clone to the new model file name
            Call FileCopy(IN_DIRECTORY & udtClone.FileName, IN_DIRECTORY &
strNewFN)
30         ' make a copy of the parent's model file for this child
            Call FileCopy(ModelFileName(IN_DIRECTORY &
ModelEmbedKey(udtClone.FileName, strKey)), _
                ModelFileName(IN_DIRECTORY & strNewFN))
            ' add the child's model to the model collection. "Thaw" the child.
35         Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & strNewFN, _
            mudtFam.ItemType)
            udtM.IsFrozen = False
            ' reset the clone index of the child
            udtM.LastClone = 0
40         ' save it
            udtM.WriteModel
            ' tell 'em about it
            Call MsgBox("Variant " & udtClone.FileName & " has been copied to " &
strNewFN, _
45         vbInformation, "Model Created")

```



```
        End If
    Next intI
End If
End With
```

```
5      UpdateTab0ControlStates
      UpdateTab2ControlStates
```

```
End Sub
```

```
Private Sub mnuFileNew_Click()
```

```
10      Dim udtWAPI As New Win32API
      Dim strFN As String
      Dim udtProgram As Program
      Dim udtItemType As ItemType
      Dim udtProximity As Proximity
      Dim blnGeneric As Boolean
15      Dim udtIni As New IniFile

      ' clear out everything
      ClearControls

      ' get family values (pun intended)
20      frmNew.Show vbModal
      If frmNew.OK = False Then GoTo Cancel

      udtProgram = frmNew.Program
      udtItemType = frmNew.ItemType
      udtProximity = frmNew.Proximity
25      blnGeneric = frmNew.Generic

      With cdlCD
          .InitDir = IN_DIRECTORY
          .FileName = ""
          .DialogTitle = "Save new family as"
30          .Filter = "Model Doc Files (*$R.doc)|*$R.doc|"
          .DefaultExt = ".doc"
          .Flags = cdIOFNHideReadOnly
          On Error GoTo Cancel
          .ShowSave
35          On Error GoTo 0
          strFN = .FileName
      End With
```

```
' see if an FN was entered
```

```
If Len(strFN) = 0 Then
```

```
    Beep
```

```
    GoTo Cancel
```

```
End If
```

```
strFN = UCase(strFN)
```

```
' don't allow family to be created if it's not in the "IN" directory
```

```
If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
```

```
    ' do nothing
```

```
Else
```

```
    Call MsgBox("Family must be located in " & IN_DIRECTORY, _  
        vbExclamation, "Error")
```

```
    GoTo Cancel
```

```
End If
```

```
' check the extension
```

```
If (InStr(1, strFN, ".doc", vbTextCompare)) = 0 Then
```

```
    Call MsgBox("Invalid file name extension.", vbExclamation, "Error")
```

```
    GoTo Cancel
```

```
End If
```

```
Dim varI As Variant
```

```
' embed $R into FN if the user hasn't
```

```
If InStr(1, strFN, "$R.doc", vbTextCompare) = 0 Then
```

```
    varI = InStr(1, strFN, ".doc", vbTextCompare)
```

```
    strFN = Mid(strFN, 1, varI - 1) & "$R.doc"
```

```
End If
```

```
' check for unique FN
```

```
If udtWAPI.FileExists(strFN) Then
```

```
    Call MsgBox("File name " & _  
        ExtractFileName(strFN) & " is not unique.", _
```

```
        vbExclamation, "Error")
```

```
    GoTo Cancel
```

```
End If
```

```
Dim strShortFN As String
```

```
strShortFN = ExtractFileName(strFN)
```

```
' create a new family object
```

```
Set mudtFam = New Family
```

```
' set file name, program, and item type
mudtFam.FileName = strFN
mudtFam.Program = udtProgram
mudtFam.ItemType = udtItemType
5 mudtFam.Proximity = udtProximity
mudtFam.Generic = blnGeneric
mudtFam.IsDirty = True
```

```
' put the family name on the status bar
stbS.Panels(pnFamilyName) = strShortFN
```

```
10 ' fill in the rest of the status bar
UpdateFamilyAttributes
```

```
' format tab 2
Call FormatTab2(mudtFam.ItemType)
```

```
' copy correct Word template to new model FN
15 Select Case mudtFam.ItemType
```

```
    Case ptStandardMC
        FileCopy App.path & "\TCASMC.doc", strFN
```

```
    Case ptQuantComp
        FileCopy App.path & "\TCAQC.doc", strFN
```

```
20    Case ptDataSuff
        FileCopy App.path & "\TCADS.doc", strFN
```

```
End Select
```

```
Dim nodN As Node
```

```
' clear out the treeview box
25 treModels.Nodes.Clear
```

```
' add the new root
Set nodN = treModels.Nodes.Add(, "R", strShortFN, imSun)
nodN.Expanded = True
nodN.sorted = True
30 nodN.Selected = True
```

```
Call mudtFam.Models.AddNew(strFN, mudtFam.ItemType)
```

```
' enable attributes button
```

```
cmdSetAttributes.Enabled = True
```

```
' force event to set active model  
treModels_Click
```

```
Cancel:
```

```
5 UpdateTab0ControlStates
```

```
Exit Sub
```

```
End Sub
```

```
Private Sub mnuFileOpen_Click()
```

```
Dim strFN As String
```

```
10
```

```
' clear out everything  
ClearControls
```

```
With cdlCD
```

```
.InitDir = IN_DIRECTORY
```

```
15
```

```
.FileName = ""
```

```
.CancelError = True
```

```
.DialogTitle = "Open model root"
```

```
.Filter = "Model Doc Files (*$R.doc)|*$R.doc|"
```

```
.DefaultExt = ".doc"
```

```
20
```

```
.Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly
```

```
On Error GoTo Cancel
```

```
.ShowOpen
```

```
On Error GoTo 0
```

```
strFN = .FileName
```

```
25
```

```
End With
```

```
' exit if there's no file name
```

```
If Len(strFN) = 0 Then
```

```
Exit Sub
```

```
End If
```

```
30
```

```
strFN = UCase(strFN)
```

```
' don't allow family to be opened if it's not in the "IN" directory
```

```
If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
```

```
' do nothing
```

```
Else
```

```

    Call MsgBox("Family must be located in " & IN_DIRECTORY, _
        vbExclamation, "Error")
    Exit Sub
End If

```

```

5      ' find all of the children
    Dim nodN As Node
    Dim strIndex As String
    Dim strT As String
    Dim varI1 As Variant
10     Dim udtWAPI As New Win32API
    Dim strNewFN As String
    Dim colFN As Collection

    ' add a wild card to the file name
    varI1 = InStr(1, strFN, ".")
15     strNewFN = Mid(strFN, 1, varI1 - 1) & "*" & Mid(strFN, varI1, _
        Len(strFN) - varI1 + 1)

    ' get a collection of file names (*.doc) matching the wild card
    Set colFN = udtWAPI.FindAllFiles(strNewFN)

    ' create a new family object
20     Set mudtFam = New Family

    Dim strMdfFN As String

    ' make sure the .mdf file is there.
    strMdfFN = left(strFN, Len(strFN) - 3) & ".mdf"
    If udtWAPI.FileExists(strMdfFN) = False Then
25         Call MsgBox("This family has a " & _
            "missing mdf file and cannot be loaded. " & _
            "File " & strMdfFN & " is not in the IN directory.", _
            vbExclamation, "Error")
        Exit Sub
30     End If

    ' set the file name of the family, read.
    mudtFam.FileName = strFN
    mudtFam.ReadFamily

    Dim udtClone As Clone

35     ' verify that all variants referenced in the family object are in
    ' the IN directory.

```

```

For Each udtClone In mudtFam.Clones
    ' the next line allows families to be renamed between TCA sessions
    udtClone.FileName = ExtractFamilyName(strFN) & _
        ExtractFamilyKey(udtClone.FileName) & ".doc"
5    If udtWAPI.FileExists(IN_DIRECTORY & udtClone.FileName) = False Then
        Call MsgBox("This family has at least " & _
            "one missing variant file and cannot be loaded. " & _
            "File " & udtClone.FileName & " is not in the IN directory.", _
            vbExclamation, "Error")
10    Exit Sub
    End If
Next udtClone

' put family name on status bar
stbS.Panels(pnFamilyName) = ExtractFileName(strFN)

15    ' format tab 2
    Call FormatTab2(mudtFam.ItemType)

    ' update the accepted listbox with leftover clones
    For Each udtClone In mudtFam.Clones
        With lstAccepted
            If udtClone.IsRouted Then
20                Call .AddItem(udtClone.FileName & ": Routed to TCS")
            Else
                Call .AddItem(udtClone.FileName)
            End If
25            .ItemData(.ListCount - 1) = udtClone.index
        End With
    Next udtClone

    ' select the first entry, if there is one
    If lstAccepted.ListCount > 0 Then
30        lstAccepted.Selected(0) = True
    End If

    ' display attribute info on status bar
    UpdateFamilyAttributes

    ' clear out the dummy list box
35    Call lstDummy.Clear

    Dim varFN As Variant
    Dim udtM As Model
    Dim intI As Integer

```

Dim intIcon As Integer

' dump the file names into a dummy list box which will sort them automatically.
' the tree control must add them in heirarchical order.

For Each varFN In colFN

varI1 = InStr(1, varFN, ".")

If IsNumeric(Mid(varFN, varI1 - 1, 1)) = False Then ' it's not a clone

Call lstDummy.AddItem(varFN) ' add the model

End If

Next varFN

Dim strMdlFN As String

For intI = 0 To lstDummy.ListCount - 1

varFN = lstDummy.List(intI)

strIndex = ModelKey(varFN)

If UCase(strIndex) = "R" Then

Set nodN = treModels.Nodes.Add(, strIndex, varFN)

Set treModels.SelectedItem = nodN

Else

Set nodN = treModels.Nodes.Add(left(strIndex, Len(strIndex) - 1), _
tvwChild, strIndex, varFN)

End If

' test to see if corresponding .mdl file exists

strMdlFN = left(varFN, Len(varFN) - 3) & ".mdl"

If udtWAPI.FileExists(strMdlFN) = False Then

Call MsgBox("This family has at least " & _

"one missing mdl file and cannot be loaded. " & _

"File " & strMdlFN & " is not in the IN directory.", _

vbExclamation, "Error")

ClearControls

Exit Sub

End If

' add a new model to the collection

Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & varFN, _
mudtFam.ItemType)

If udtM.IsFrozen Then

nodN.Image = imSnowflake

Else

nodN.Image = imSun

End If

nodN.Expanded = True

nodN.sorted = True

Next intI

' enable attributes button
cmdSetAttributes.Enabled = True

' force event to set active model
5 treModels_Click

Cancel:

UpdateTab0ControlStates

Exit Sub

End Sub

10 Private Sub mnuFileImportItem_Click()

Dim udtIni As New IniFile
Dim strFN As String

' clear out everything
15 ClearControls

With cdlCD

.InitDir = IN_DIRECTORY

.FileName = ""

.CancelError = True

20 .DialogTitle = "Open locked item"

.Filter = "Item Doc Files (*.doc)|*.doc|"

.DefaultExt = ".doc"

.Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly

On Error GoTo Cancel

25 .ShowOpen

On Error GoTo 0

strFN = .FileName

End With

' End If

30 ' exit if there's no file name

If Len(strFN) = 0 Then

Exit Sub

End If

' don't allow locked item to be opened if it's not in the "IN" directory


```
If InStr(1, strFN, IN_DIRECTORY, vbTextCompare) Then
```

```
    ' do nothing
```

```
Else
```

```
    Call MsgBox("Locked item must be located in " & IN_DIRECTORY, _  
vbExclamation, "Error")
```

```
    Exit Sub
```

```
End If
```

```
' set the FN of the ini that accompanies the locked item
```

```
udtIni.FN = IN_DIRECTORY & ExtractFileNameNoExt(strFN) & ".ini"
```

```
Dim udtW As New Win32API
```

```
If udtW.FileExists(udtIni.FN) = False Then
```

```
    Call MsgBox("Ini file must accompany locked item " & ExtractFileName(strFN) & _  
".", vbExclamation, "Error")
```

```
    Exit Sub
```

```
End If
```

```
Dim udtProgram As Program
```

```
Dim udtDeliveryMode As DeliveryMode
```

```
Dim udtItemType As ItemType
```

```
Dim strAccNum As String
```

```
' find out about this locked item from the .ini file
```

```
Select Case udtIni.GetProfileString("LockedItemData", "Program")
```

```
    Case "GRE"
```

```
        udtProgram = prGRE
```

```
    Case "GMAT"
```

```
        udtProgram = prGMAT
```

```
    Case "SAT"
```

```
        udtProgram = prSAT
```

```
    Case "Not Found"
```

```
        Call MsgBox("No Program entry found in ini file " & ExtractFileName(strFN) & _  
".", vbExclamation, "Error")
```

```
        Exit Sub
```

```
End Select
```

```
Select Case udtIni.GetProfileString("LockedItemData", "DeliveryMode")
```

```
    Case "CBT"
```

```
        udtDeliveryMode = dmCBT
```

```
    Case "PPT"
```

```
        udtDeliveryMode = dmPPT
```

```
    Case "Not Found"
```

```
        Call MsgBox("No DeliveryMode entry found in ini file " & ExtractFileName(strFN) & _
```

```

        ".", vbExclamation, "Error")
    Exit Sub
End Select

```

```

Select Case udtIni.GetProfileString("LockedItemData", "ItemType")
5   Case "MC Item", "QantDisc", "MC", "Multiple Choice"
        udtItemType = ptStandardMC
    Case "DataSuff", "DS", "Data Sufficiency"
        udtItemType = ptDataSuff
    Case "QC Discrete", "QantComp", "QC", "Quantitative Comparison"
10   udtItemType = ptQuantComp
    Case "Not Found"
        Call MsgBox("No ItemType entry found in ini file " & ExtractFileName(strFN) & _
            ".", vbExclamation, "Error")
        Exit Sub
15 End Select

```

```

strAccNum = udtIni.GetProfileString("LockedItemData", "LockedAccnum")
If strAccNum = "Not Found" Then strAccNum = ""

```

```

' initialize locked item object
Dim udtLI As New LockedItem

```

```

20 udtLI.LockedItemFileName = strFN
    udtLI.WordInstance = mudtWord

```

```

If udtLI.OpenLockedItemDoc = False Then ' we couldn't figure out what doc and item type it
was

```

```

    Call MsgBox("Locked item file appears to be damaged.", vbExclamation, "Error")
    udtLI.CloseLockedItemDoc
    Exit Sub
End If

```

```

With cdlCD
    .FileName = ""
30   .DialogTitle = "Save new family based on this locked item as"
    .Filter = "Model Doc Files (*.doc)|*.doc|"
    .DefaultExt = ".doc"
    .Flags = cdlOFNHideReadOnly
    On Error GoTo CloseAndCancel
35   .ShowSave
    On Error GoTo 0
    strFN = .FileName
End With
' End If

```

' see if an FN was entered

If Len(strFN) = 0 Then

 Beep

 Exit Sub

End If

strFN = UCase(strFN)

' check the extension

If (InStr(1, strFN, ".doc", vbTextCompare)) = 0 Then

 Call MsgBox("Invalid file name extension.", vbExclamation, "Error")

 Exit Sub

End If

Dim varI As Variant

' embed \$R into FN if the user hasn't

If InStr(1, strFN, "\$R.doc", vbTextCompare) = 0 Then

 varI = InStr(1, strFN, ".doc", vbTextCompare)

 strFN = Mid(strFN, 1, varI - 1) & "\$R.doc"

End If

' check for unique FN

Dim udtWAPI As New Win32API

If udtWAPI.FileExists(strFN) Then

 Call MsgBox("File name " & _
 ExtractFileName(strFN) & " is not unique.", _
 vbExclamation, "Error")

 Exit Sub

End If

' copy the ini file of the locked item to the family name

Call FileCopy(udtIni.FN, left(strFN, Len(strFN) - 3) & ".ini")

Dim strShortFN As String

strShortFN = ExtractFileName(strFN)

' create a new family object

Set mudtFam = New Family

' put family name on status bar

stbS.Panels(pnFamilyName) = strShortFN

' set file name, program, and item type

```
mudtFam.FileName = strFN
mudtFam.Program = udtProgram
mudtFam.ItemType = udtItemType
mudtFam.AccNum = strAccNum
mudtFam.IsDirty = True
```

```
' format tab 2
Call FormatTab2(mudtFam.ItemType)
```

```
' copy correct Word template to new model FN
Select Case mudtFam.ItemType
```

```
    Case ptStandardMC
        FileCopy App.path & "\TCASMC.doc", strFN
```

```
    Case ptQuantComp
        FileCopy App.path & "\TCAQC.doc", strFN
```

```
    Case ptDataSuff
        FileCopy App.path & "\TCADS.doc", strFN
```

```
End Select
```

```
Dim nodN As Node
```

```
' clear out the treeview box
treModels.Nodes.Clear
```

```
' add the new root
Set nodN = treModels.Nodes.Add(, "R", strShortFN, imSun)
nodN.Expanded = True
nodN.sorted = True
nodN.Selected = True
```

```
Call mudtFam.Models.AddNew(strFN, mudtFam.ItemType)
```

```
mudtFam.Generic = False
mudtFam.Proximity = prNear
```

```
' enable attributes button
cmdSetAttributes.Enabled = True
```

```
' force event to set attributes
cmdSetAttributes_Click
```

```
' force event to set active model
treModels_Click
```

```
5      Select Case udtItemType
        Case ptStandardMC
          Select Case udtDeliveryMode
            Case dmCBT
              Call udtLI.ConvertCBTSMCItem
            Case dmPPT
              Call udtLI.ConvertPPTSMCItem
10         End Select
        Case ptDataSuff
          Call udtLI.ConvertDSItem
        Case ptQuantComp
          Select Case udtDeliveryMode
15             Case dmCBT
              Call udtLI.ConvertCBTQCItem
            Case dmPPT
              Call udtLI.ConvertPPTQCItem
          End Select
20      End Select
```

```
CloseAndCancel:
```

```
    udtLI.CloseLockedItemDoc
```

```
Cancel:
```

```
    UpdateTab0ControlStates
```

```
25    Exit Sub
```

```
End Sub
```

```
Private Sub mnuFileExit_Click()
```

```
    Call Form_Unload(0)
    End
```

```
30 End Sub
```

```
'Private Sub ReturnToTab0()
```

```
'
'   Dim intPrevTab As Integer
'
```

```

' If sstMainTab.Tab = 0 Then Exit Sub
'
' intPrevTab = sstMainTab.Tab
' sstMainTab.Tab = 0
5 ' Call sstMainTab_Click(intPrevTab)
'
'End Sub
'
Private Sub mnuFilePrintSetup_Click()
10
    cdlCD.Flags = cdlPDPrintSetup

    On Error GoTo Cancel
    cdlCD.ShowPrinter
15    On Error GoTo 0

Cancel:

    Exit Sub

End Sub

20 Private Sub mnuHelpAbout_Click()

    frmAbout.Show vbModal

End Sub

Private Sub mnuTreeExtend_Click()

25    Dim nodN As Node
    Dim strFN As String
    Dim strNewFN As String
    Dim strKey As String
    Dim strT As String
    Dim strNewKey As String

30    If treModels.SelectedItem Is Nothing Then Exit Sub

    Set nodN = treModels.SelectedItem
    strFN = nodN.Text

    ' confirm this operation
    If MsgBox("Make a child model from model " & strFN & "?", _
35        vbQuestion + vbYesNo, "Confirm") = vbNo Then

```

Exit Sub
End If

strKey = ModelKey(strFN)

strNewKey = NextModelKey(strFN)

5 ' add the child to the tree
strNewFN = ModelEmbedKey(strFN, strNewKey)
Set nodN = treModels.Nodes.Add(strKey, tvwChild, strNewKey, strNewFN)
nodN.Expanded = True
nodN.sorted = True
10 nodN.Image = imSun

' deactivate active model to close it before file copies, if the active
' model is being extended.

Dim blnReopenModel As Boolean

blnReopenModel = False
15 If strFN = stbS.Panels(pnActiveModelName) Then
Call mudtFam.ActiveModel.CloseDoc
blnReopenModel = True
End If

' make a copy of the parent's word doc for this child
20 Call FileCopy(IN_DIRECTORY & strFN, IN_DIRECTORY & strNewFN)

' make a copy of the parent's model file for this child
Call FileCopy(IN_DIRECTORY & ModelFileName(strFN), IN_DIRECTORY &
ModelFileName(strNewFN))

' add the child's model to the model collection. "Thaw" the child.
25 Dim udtM As Model
Set udtM = mudtFam.Models.AddExisting(IN_DIRECTORY & strNewFN, _
mudtFam.ItemType)
udtM.IsFrozen = False

' reset the clone index of the child
30 udtM.LastClone = 0

' reset the checksums
udtM.InitChecksums

' save it

udtM.WriteModel

If blnReopenModel Then

 Call mudtFam.ActiveModel.OpenDoc(mudtWord)

End If

5 End Sub

Private Sub mnuTreeRemove_Click()

 Dim nodN As Node

 Dim strFN As String

 Dim strKey As String

10 If treModels.SelectedItem Is Nothing Then Exit Sub

 Set nodN = treModels.SelectedItem

 strFN = nodN.Text

 strKey = ModelKey(strFN)

 Dim colIndices As New Collection

15 ' don't remove if this node or any descendant nodes are frozen
 Dim udtModel As Model

 ' check selected node

 If treModels.SelectedItem.index = 1 Then ' it's the root model

 Call MsgBox("The root model can't be removed.", vbExclamation, "Error")

20 Exit Sub

End If

 Set udtModel = mudtFam.Models.Item(treModels.SelectedItem)

 If udtModel.IsFrozen Then

 Call MsgBox("Can't remove frozen model.", vbExclamation, "Error")

25 Exit Sub

 Else

 Call colIndices.Add(treModels.SelectedItem.index)

 End If

 Dim blnDone As Boolean

30 blnDone = False

 ' check if any of it's descendants are frozen

Do


```

Set nodN = nodN.Child
If nodN Is Nothing Then
    ' do nothing
Else
5     Do
        If mudtFam.Models.Item(nodN.Text).IsFrozen Then
            Call MsgBox("Can't remove model with one or more frozen descendants.", _
                vbExclamation, "Error")
            Exit Sub
10        End If
        Call colIndices.Add(nodN.index)
        Loop Until nodN.index = nodN.LastSibling.index
    End If
    Loop Until nodN Is Nothing

15    ' confirm this operation
    If MsgBox("Remove model " & strFN & " and it's children?", _
        vbQuestion + vbYesNo, "Confirm") = vbNo Then
        Exit Sub
    End If

20    ' close active model document as we're deleting it
    mudtFam.ActiveModel.CloseDoc

    mudtFam.ActiveModel = Nothing
    stbS.Panels(pnActiveModelIcon).Picture = Nothing
    stbS.Panels(pnActiveModelName) = ""

25    Dim varIndex As Variant

    ' remove all effected models from the family
    For Each varIndex In colIndices
        Call mudtFam.Models.Remove(treModels.Nodes(varIndex))
        Kill IN_DIRECTORY & left(treModels.Nodes(varIndex), _
30        Len(treModels.Nodes(varIndex)) - 3) & "*"
    Next varIndex

    ' remove them from the tree control
    Call treModels.Nodes.Remove(colIndices(1))

End Sub

35 Private Sub mnuVariablesAdd_Click()

    frmVariable.AddEditFlag = aeAdd

```

End Sub

Private Sub mnuVariablesEdit_Click()

frmVariable.AddEditFlag = aeEdit

End Sub

5 Private Sub mnuVariablesRemove_Click()

Dim intInd As Integer

intInd = lstVariables.ListIndex ' Get index

' Make sure list item is selected

If intInd < 0 Then

10 Beep

Exit Sub

End If

Dim strVN As String

strVN = mudtFam.ActiveModel.Variables.Item(Str(lstVariables.ItemData(intInd))).name

' confirm this operation

If MsgBox("Remove variable " & strVN & "?", _
vbQuestion + vbYesNo, "Confirm") = vbNo Then

Exit Sub

End If

' Remove the variable from the collection using the key in the list box

20 Call mudtFam.ActiveModel.Variables.Remove(Str(lstVariables.ItemData(intInd)))

' Remove the variable from the list box

Call lstVariables.RemoveItem(intInd)

UpdateTab1ControlStates

25 End Sub

'Empty the variable list box

Private Sub mnuVariablesRemoveAll_Click()

' confirm this operation

If MsgBox("Remove all variables?", _

```
vbQuestion + vbYesNo, "Confirm") = vbNo Then
Exit Sub
End If
```

```
'clear the list box
lstVariables.Clear
```

```
' empty the collection
mudtFam.ActiveModel.Variables.Clear
```

```
UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub mnuVariablesEnableAll_Click()
```

```
Call SetAllCheckboxes(True)
```

```
UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub mnuVariablesDisableAll_Click()
```

```
Call SetAllCheckboxes(False)
```

```
UpdateTab1ControlStates
```

```
End Sub
```

```
Private Sub mnuVariablesTest_Click()
```

```
Call TestConstraints(tcTestVariables)
```

```
End Sub
```

```
Private Sub mnuConstraintsAdd_Click()
```

```
' set the add flag for frmConstraints
frmConstraints.AddEditFlag = aeAdd
```

```
' set the list box
```

```
frmConstraints.ListBox = lstConstraints(mintConstrLBInd)
```

```
' set the model
```

```
frmConstraints.Model = mudtFam.ActiveModel
```

```
' set the constraint type
```

```
frmConstraints.ConstraintType = mintConstrLBInd
```

```
' crank up the form
frmConstraints.Show vbModal
```

```
Call UpdateTab1ControlStates(mintConstrLBInd)
```

```
End Sub
```

```
5 Private Sub mnuConstraintsEdit_Click()
```

```
    If lstConstraints(mintConstrLBInd).ListIndex >= 0 Then ' Make sure list item is selected
```

```
        ' set the edit flag for frmConstraints
        frmConstraints.AddEditFlag = aeEdit
```

```
        ' set the list box
```

```
10    frmConstraints.ListBox = lstConstraints(mintConstrLBInd)
```

```
        ' set the model
```

```
        frmConstraints.Model = mudtFam.ActiveModel
```

```
        ' set the constraint
```

```
        With lstConstraints(mintConstrLBInd)
```

```
15            frmConstraints.Constraint = _
                mudtFam.ActiveModel.Constraints.Item(Str(.ItemData(.ListIndex)))
```

```
        End With
```

```
        ' set the constraint type
```

```
        frmConstraints.ConstraintType = mintConstrLBInd
```

```
20    ' crank up the form
```

```
        frmConstraints.Show vbModal
```

```
Else
```

```
    Beep
```

```
End If
```

```
25    Call UpdateTab1ControlStates(mintConstrLBInd)
```

```
End Sub
```

```
Private Sub mnuConstraintsRemove_Click()
```

```
    Dim intInd As Integer
```

```
    intInd = lstConstraints(mintConstrLBInd).ListIndex ' Get index
```

```
30    ' Make sure list item is selected
```

```
    If intInd < 0 Then
```

```
        Beep
```

```
        Exit Sub
```

```
    End If
```

```
Dim udtCon As Constraint
```

```
Set udtCon = _
```

```
5 mudtFam.ActiveModel.Constraints.Item(Str(lstConstraints(mintConstrLBInd).ItemData(intInd))  
)
```

```
' confirm this operation
```

```
If MsgBox("Remove constraint " & udtCon.ConstraintString & "?", _  
vbQuestion + vbYesNo, "Confirm") = vbNo Then
```

```
Exit Sub
```

```
10 End If
```

```
' Remove the variable from the collection using the key in the list box
```

```
Call
```

```
mudtFam.ActiveModel.Constraints.Remove(Str(lstConstraints(mintConstrLBInd).ItemData(intInd)))
```

```
15 ' Remove the variable from the list box
```

```
Call lstConstraints(mintConstrLBInd).RemoveItem(intInd)
```

```
Call UpdateTab1ControlStates(mintConstrLBInd)
```

```
End Sub
```

```
Private Sub mnuConstraintsRemoveAll_Click()
```

```
20 ' confirm this operation
```

```
If MsgBox("Remove all constraints in this list box?", _  
vbQuestion + vbYesNo, "Confirm") = vbNo Then
```

```
Exit Sub
```

```
End If
```

```
25 'clear the list box
```

```
lstConstraints(mintConstrLBInd).Clear
```

```
' empty the collection
```

```
Call mudtFam.ActiveModel.Constraints.Clear(mintConstrLBInd)
```

```
Call UpdateTab1ControlStates(mintConstrLBInd)
```

```
30 End Sub
```

```
Private Sub mnuConstraintsEnableAll_Click()
```

```
Call SetAllCheckboxes(True)
```

Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsDisableAll_Click()

Call SetAllCheckboxes(False)

5 Call UpdateTab1ControlStates(mintConstrLBInd)

End Sub

Private Sub mnuConstraintsTest_Click()

cmdSaveModel_Click ' force a save

Select Case mintConstrLBInd

10 Case ctVariation
Call TestConstraints(tcTestVariationConstraints)

Case ctDistractor
Call TestConstraints(tcTestDistractorConstraints)

End Select

15 End Sub

Private Sub mnuAcceptedProfile_Click()

Dim udtClone As Clone

Dim intI As Integer

' set the family

20 frmDifficulty.Family = mudtFam

' set the clone

With lstAccepted

For intI = 0 To .ListCount - 1

If .Selected(intI) Then

25 Set udtClone = _
mudtFam.Clones.Item(Str(.ItemData(intI)))
frmDifficulty.Clone = udtClone

Exit For

End If

30 Next intI

End With

```
' give frmDifficulty a caption
frmDifficulty.Caption = "Profile of variant " & _
    ExtractFileName(udtClone.FileName)
```

```
5      ' crank up the form
      frmDifficulty.Show vbModal
```

```
      If udtClone.IsRouted Then
          lstAccepted.List(intI) = udtClone.FileName & ": Routed to TCS"
      Else
          lstAccepted.List(intI) = udtClone.FileName
10     End If
```

```
End Sub
```

```
Private Sub mnuAcceptedCopy_Click()
```

```
    Dim udtClone As Clone
```

```
    ' this menu option is only active if a variant with a completed profile
    ' is currently selected.
```

```
    With lstAccepted
```

```
        Set udtClone = mudtFam.Clones.Item(Str(.ItemData(.ListIndex)))
```

```
    End With
```

```
    ' copy necessary stuff into a holding area
```

```
    Set mudtClone = udtClone
```

```
    UpdateTab0ControlStates
```

```
End Sub
```

```
' this menu option is only active if a profile has been copied
```

```
Private Sub mnuAcceptedPaste_Click()
```

```
25     Dim udtClone As Clone
        Dim intI As Integer
```

```
    With lstAccepted
```

```
        If .SelCount > 0 Then
```

```
            ' confirm this operation
```

```
30            If MsgBox("Paste profile of variant " & mudtClone.FileName & _
                " to all selected variants?", _
                vbQuestion + vbYesNo, "Confirm") = vbNo Then
                Exit Sub
```

```

End If
For intI = 0 To .ListCount - 1
    If .Selected(intI) Then
        Set udtClone = mudtFam.Clones.Item(Str(.ItemData(intI)))
        ' copy necessary stuff from the holding area
        udtClone.Domain = mudtClone.Domain
        udtClone.BatchID = mudtClone.BatchID
        udtClone.DeliveryMode = mudtClone.DeliveryMode
        udtClone.Nature = mudtClone.Nature
        udtClone.IsRouted = mudtClone.IsRouted
        udtClone.TDEstimate = mudtClone.TDEstimate
        udtClone.IsDifficultyCalculated = mudtClone.IsDifficultyCalculated
        If udtClone.IsDifficultyCalculated Then
            udtClone.DiffEst = mudtClone.DiffEst.Copy
        End If
        If udtClone.IsRouted Then
            .List(intI) = udtClone.FileName & ": Routed to TCS"
        Else
            .List(intI) = udtClone.FileName
        End If
    End If
Next intI
End If
End With

End Sub

' checks/unchecks all checkboxes in a listbox and enable/disable their
' associated variable or constraint objects

Private Sub SetAllCheckboxes(ByVal blnBool As Boolean)

    Dim i As Integer

    For i = 0 To (mlstCurrentListBox.ListCount - 1)
        mlstCurrentListBox.Selected(i) = blnBool
    Next i

    Dim udtV As Variable
    Dim udtC As Constraint

    If mlstCurrentListBox.name = "lstVariables" Then
        For Each udtV In mudtFam.ActiveModel.Variables
            udtV.Enabled = blnBool
        Next udtV
    End If

```



```

Else
    For i = 0 To (mlstCurrentListBox.ListCount - 1)
        Set udtC =
mudtFam.ActiveModel.Constraints.Item(Str(mlstCurrentListBox.ItemData(i)))
5        udtC.Enabled = blnBool
    Next i
End If

End Sub

Private Sub mwudtModelTest_PrologFinished()

10 End Sub

Private Sub sstMainTab_Click(PreviousTab As Integer)

    Static blnRecurring As Boolean
    Static bytMessage As Byte

    If blnRecurring Then
15        Select Case bytMessage
            Case 1
                Call MsgBox("Open a model family using the File menu.", _
                    vbExclamation, "Error")
            Case 2
20                Call MsgBox("Set the active model by clicking on a model.", _
                    vbExclamation, "Error")
        End Select
        blnRecurring = False
        Exit Sub
25    End If

    ' error conditions
    If sstMainTab.Tab > 0 Then
        If treModels.Nodes.Count = 0 Then ' family hasn't been set
            bytMessage = 1
30            blnRecurring = True
            sstMainTab.Tab = PreviousTab ' will trigger recursion
            Exit Sub
        End If
    End If

35    If sstMainTab.Tab = 1 Or sstMainTab.Tab = 2 Then
        If mudtFam.ActiveModel Is Nothing Then ' active model has not been set
            bytMessage = 2

```

```

        blnRecurring = True
        sstMainTab.Tab = PreviousTab ' will trigger recursion
        Exit Sub
    End If
5    End If

    ' if we got here, everything's ok!
    If PreviousTab = 2 Then
        txtNum2Generate = ""
    End If

10    If PreviousTab = 1 Then
        If mudtFam.ActiveModel.IsDirty Then
            KillVariants 'delete any variants on tab 3
            mudtFam.ActiveModel.InitTempChecksums ' initialize temp checksums
        End If
15    End If

    ' save family
    mudtFam.WriteFamily

    ' save the active model
    If mudtFam.ActiveModel Is Nothing Then
20        ' do nothing
    Else
        mudtFam.ActiveModel.WriteModel
    End If

    Select Case sstMainTab.Tab

25        Case 0
            ' enable new/open
            cmdSetAttributes.Default = True
            mnuFileNew.Enabled = True
            mnuFileOpen.Enabled = True
            mnuFileImportItem.Enabled = True
30            If PreviousTab = 2 Then
                mudtFam.ActiveModel.CloseAllCloneDocs
                Call mudtFam.ActiveModel.OpenDoc(mudtWord)
            End If

35            ' if there are no variants, disable the print button
            If lstAccepted.ListCount > 0 Then
                cmdPrintBatch.Enabled = True
            Else

```

```
cmdPrintBatch.Enabled = False
End If
```

```
Case 1
```

```
cmdSaveModel.Default = True
' disable new/open
mnuFileNew.Enabled = False
mnuFileOpen.Enabled = False
mnuFileImportItem.Enabled = False
' warn if variants exist in lstDisposition and model isn't frozen
If mudtFam.ActiveModel.IsFrozen = False Then
    If lstDisposition.ListCount > 0 Then ' variants exist
        Call MsgBox("Variants on tab 3 will be deleted if " & _
            "the model is changed.", vbInformation, "Warning")
    End If
End If
If PreviousTab = 0 Then
    mudtFam.CloseAllCloneDocs
    Call mudtFam.ActiveModel.OpenDoc(mudtWord)
End If
If PreviousTab = 2 Then
    mudtFam.ActiveModel.CloseAllCloneDocs
    Call mudtFam.ActiveModel.OpenDoc(mudtWord)
End If
```

```
Case 2
```

```
cmdGenerate.Default = True
' disable new/open
mnuFileNew.Enabled = False
mnuFileOpen.Enabled = False
mnuFileImportItem.Enabled = False

' disable the generate button
cmdGenerate.Enabled = False

' if there are no variants, disable the print button
If lstDisposition.ListCount > 0 Then
    cmdPrintVariants.Enabled = True
Else
    cmdPrintVariants.Enabled = False
End If

If PreviousTab = 0 Then
    mudtFam.CloseAllCloneDocs
End If
```

```

        ' display the currently selected document
        With lstDisposition
            If .ListCount > 0 Then ' a valid selection has been made
                Call mudtFam.ActiveModel.Clones.Item _
5                (Str(.ItemData(.ListIndex))).OpenDoc(mudtWord, IN_DIRECTORY)
            Else
                Call mudtFam.ActiveModel.OpenDoc(mudtWord)
            End If
        End With

10    End Select

End Sub

' restore full window drag, if necessary
Private Sub sstMainTab_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)

15    Dim udtW As Win32API

    If mbInRestoreFullWindowDrag Then
        Set udtW = New Win32API
        udtW.TurnOnFullWindowDrag
        mbInRestoreFullWindowDrag = False
20    End If

    If mudtWord Is Nothing Then Exit Sub

    If sstMainTab.Tab = 1 Then ' do this first, as there will be an active doc
        ' on tab 1
        If mudtWord.WordApp.ActiveDocument.Saved = False And _
25        cmdSaveModel.Enabled = False Then
            If Not mudtFam.ActiveModel.IsFrozen Then
                mudtFam.ActiveModel.IsDirty = True
                UpdateTab1ControlStates
            End If
        End If
30    End If
End If

End Sub

Private Sub treModels_Click()

    Dim nodN As Node

```

If treModels.SelectedItem Is Nothing Then Exit Sub

Set nodN = treModels.SelectedItem

' put model icon and name on status bar

stbS.Panels(pnActiveModelIcon).Picture = imLI.ListImages(nodN.Image).Picture

stbS.Panels(pnActiveModelName) = treModels.SelectedItem

' close doc for existing active model

If mudtFam.ActiveModel Is Nothing Then

 ' do nothing

Else

 mudtFam.ActiveModel.CloseDoc

End If

' set the new active model and activate it

mudtFam.ActiveModel = mudtFam.Models.Item(treModels.SelectedItem)

Call mudtFam.ActiveModel.OpenDoc(mudtWord)

' clear out the Variable list box

lstVariables.Clear

' populate the variable list box with this model's variables

Dim udtVar As Variable

For Each udtVar In mudtFam.ActiveModel.Variables

 With lstVariables

 Call .AddItem(udtVar.ScreenFormat)

 .ItemData(.ListCount - 1) = udtVar.index

 .Selected(.ListCount - 1) = udtVar.Enabled

 End With

Next udtVar

Dim intI

' clear out the constraint list boxes

lstConstraints(0).Clear

lstConstraints(1).Clear

' populate the constraint list boxes with this model's constraints

Dim udtCon As Constraint

For Each udtCon In mudtFam.ActiveModel.Constraints

 intI = udtCon.ConstraintType

 With lstConstraints(intI)

```

        Call .AddItem(udtCon.ConstraintString)
        .ItemData(.ListCount - 1) = udtCon.index
        .Selected(.ListCount - 1) = udtCon.Enabled
    End With
5    Next udtCon

    ' populate comments form
    frmComments.Comment = mudtFam.ActiveModel.Comments

    ' clear out the clone disposition list box
    lstDisposition.Clear

10    ' populate the clone list box with this model's clones
    Dim udtClone As Clone

    With lstDisposition
        For Each udtClone In mudtFam.ActiveModel.Clones
            Call .AddItem(ExtractFileName(udtClone.FileName))
            .ItemData(.ListCount - 1) = udtClone.index
15        Next udtClone
    End With

    ' save the active model
    mudtFam.ActiveModel.WriteModel

20    ' adjust menu/button states depending on active model properties
    UpdateTab1ControlStates
    UpdateTab2ControlStates

    ' enable extend
    mnuTreeExtend.Enabled = True

25 End Sub

Private Sub treModels_MouseUp(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

    If treModels.Nodes.Count > 0 Then
        If Button = vbRightButton Then
30            PopupMenu mnuTree
        End If
    End If

End Sub

```

```
Private Sub txtNum2Generate_Change()
```

```
    ' If Val(txtNum2Generate) > 0 Then  
    '     cmdGenerate.Enabled = True  
    ' Else  
5    '     cmdGenerate.Enabled = False  
    ' End If
```

```
End Sub
```

```
Private Sub txtVariablize_GotFocus()
```

```
    If mudtWord.DocumentsCount = 0 Then  
10        Beep  
    Else  
        If mudtWord.SelectionType < wdSelectionNormal Then  
            Call MsgBox("Nothing is selected.", vbExclamation, "Error")  
        Else  
15        Call AddUndefinedVariables(mudtWord.SelectionText)  
        End If  
    End If
```

```
End Sub
```

```
' scans a string for undefined variable names and add them to  
20 ' the variable collection and list box
```

```
Public Sub AddUndefinedVariables(ByVal strNames As String)
```

```
    Dim colC As Collection  
    Dim strS As Variant  
    Dim udtVar As Variable  
25    Dim colDummy As New Collection
```

```
    Set colC = UndefinedNames(strNames)
```

```
' don't do it if the model is frozen!
```

```
If Not mudtFam Is Nothing Then
```

```
    If Not mudtFam.ActiveModel Is Nothing Then
```

```
        If mudtFam.ActiveModel.IsFrozen Then
```

```
            Call MsgBox("Variables cannot be added to a frozen model.", _  
                vbExclamation, "Error")
```

```
            Exit Sub
```

```
35        End If
```

```
    End If
```

End If

For Each strS In colC

If MsgBox("Auto-define variable " & strS & "?", vbQuestion + vbYesNo, _
"New variable detected") = vbYes Then

```
5      Select Case left(strS, 1)
      Case "I"
          Set udtVar = mudtFam.ActiveModel.Variables.AddInteger(strS, _
              True, "1", "100", "1", False, True)
      Case "R"
10         Set udtVar = mudtFam.ActiveModel.Variables.AddReal(strS, _
              True, "1", "100", "1", False, True, True, ".01", True)
      Case "S"
          Set udtVar = mudtFam.ActiveModel.Variables.AddString(strS, _
              True, True, Chr(164), True, colDummy)
15         Case "F"
              Set udtVar = mudtFam.ActiveModel.Variables.AddFraction(strS, _
                  True, "1", "1", "100", "1", "1", "1", False, True, False)
      Case "U"
          Set udtVar = mudtFam.ActiveModel.Variables.AddUntyped(strS, _
20             True, False)
      Case Else ' assume untyped
          Set udtVar = mudtFam.ActiveModel.Variables.AddUntyped(strS, _
              True, False)
      End Select

25      With lstVariables
          ' Add the new variable to the variable list box
          Call .AddItem(udtVar.ScreenFormat)
          ' Set ItemData to index value of the variable object
          .ItemData(.ListCount - 1) = udtVar.index
30          ' Check the check box
          .Selected(.ListCount - 1) = True
      End With
```

End If

Next strS

```
35      ' update control states
      If colC.Count > 0 Then
          UpdateTab1ControlStates
      End If
```


End Sub

' accepts a string and parses it for undefined variable names. Returns a
' collection of the variable names that are unique.

Public Function UndefinedNames(ByVal strS As String) As Collection

```
5      Dim lngStart As Long
      Dim lngEnd As Long
      Dim strT As String
      Dim byt1 As Byte
      Dim byt2 As Byte
10     Dim colC As New Collection
      Dim blnDup As Boolean
      Dim varT As Variant

      ' parse the variable names out of strS
      For lngStart = 1 To Len(strS)
15         byt1 = Asc(Mid(strS, lngStart, 1))
         If byt1 >= 65 And byt1 <= 90 Then
             For lngEnd = lngStart + 1 To Len(strS)
                 byt2 = Asc(Mid(strS, lngEnd, 1))
                 Select Case byt2
20                     Case 48 To 57, 65 To 90, 97 To 122
                         ' if it's 0 to 9, A to Z, or a to z, continue searching
                     Case Else
                         ' if it's not, assume end of variable name has been found
                         Exit For
25                     End Select
                 Next lngEnd
                 strT = Mid(strS, lngStart, lngEnd - lngStart)
                 ' throw name away if it's already in colC
                 blnDup = False
30                 For Each varT In colC
                     If UCase(varT) = UCase(strT) Then
                         blnDup = True
                     End If
                 Next varT
35                 ' make sure name is not a Prolog function
                 If blnDup = False Then
                     ' throw name away if it's already in the main variable collection
                     If mudtFam.ActiveModel.Variables.UniqueName(strT) Then
                         Call colC.Add(strT)
40                     End If
                 End If
            End For
```

```
        lngStart = lngEnd
    End If
Next lngStart
```

```
Set UndefinedNames = colC
```

```
5 End Function
```

```
Private Sub TestConstraints(ByVal udtTestType As TestType)
```

```
    Dim strVN As String
    Dim blnUnderconstrained As Boolean
    Dim blnTestAborted As Boolean
```

```
10 If mudtFam.ActiveModel.ConstraintsOK(udtTestType, mudtProlog, _
    blnUnderconstrained, blnTestAborted, strVN) Then
    Call MsgBox("Looks good!", vbExclamation, "Test Result")
ElseIf blnTestAborted Then
15 Call MsgBox("Test aborted!", vbExclamation, "Test Result")
ElseIf blnUnderconstrained Then
    Call MsgBox("Variable " & strVN & " is underconstrained!", _
        vbExclamation, "Test Result")
Else
20 Call MsgBox("No solutions exist!", vbExclamation, "Test Result")
End If
```

```
End Sub
```

```
' displays the family attributes on the status bar
```

```
25 Private Sub UpdateFamilyAttributes()
```

```
    Select Case mudtFam.Program
    Case prGRE
        stbS.Panels(pnProgramName) = "GRE"
    Case prGMAT
30 stbS.Panels(pnProgramName) = "GMAT"
    Case prSAT
        stbS.Panels(pnProgramName) = "SAT"
    End Select
```

```
35 Select Case mudtFam.ItemType
    Case ptStandardMC
        stbS.Panels(pnItemType) = "SMC"
    Case ptQuantComp
```

```

        stbS.Panels(pnItemType) = "QC"
    Case ptDataSuff
        stbS.Panels(pnItemType) = "DS"
End Select

```

```

5   If mudtFam.Generic Then
        stbS.Panels(pnGeneric) = "Generic"
    Else
        stbS.Panels(pnGeneric) = "Non generic"
    End If

```

```

10  Select Case mudtFam.Proximity
        Case prNear
            stbS.Panels(pnProximity) = "Near"
        Case prMedium
            stbS.Panels(pnProximity) = "Medium"
15  Case prFar
            stbS.Panels(pnProximity) = "Far"
    End Select

```

```
End Sub
```

' returns the model file name given the doc file name

```
20 Private Function ModelFileName(ByVal strDocFN As String) As String
```

```

    ModelFileName = left(strDocFN, Len(strDocFN) - 4) & ".mdl"

```

```
End Function
```

' extracts the key from a model file name

```
Private Function ModelKey(ByVal strFN As String) As String
```

```

25  Dim varI1 As Variant
    Dim varI2 As Variant
    Dim intI As Integer
    Dim strS As String

```

```

    varI1 = InStr(1, strFN, "$")
30  varI2 = InStr(varI1, strFN, ".")

```

```

    ' strip off numbers or spaces to the left of the "."
    intI = varI2
    Do While intI > varI1
        intI = intI - 1
    Loop

```

```

        strS = Mid(strFN, intI, 1)
        If Asc(strS) >= 65 And Asc(strS) <= 91 Then ' it's A to Z
            varI2 = intI + 1
            Exit Do
5      End If
    Loop

```

```

    ModelKey = Mid(strFN, varI1 + 1, varI2 - varI1 - 1)

```

End Function

' embeds a new key into a model file name

```

10 Private Function ModelEmbedKey(ByVal strFN As String, ByVal strNewKey As String) _
    As String

```

```

    Dim varI1 As Variant
    Dim varI2 As Variant
    Dim intI As Integer
15 Dim strS As String

```

```

    varI1 = InStr(1, strFN, "$")
    varI2 = InStr(varI1, strFN, ".")

```

```

    ' strip off numbers or spaces to the left of the "."
    intI = varI2

```

```

20 Do While intI > varI1
    intI = intI - 1
    strS = Mid(strFN, intI, 1)
    If Asc(strS) >= 65 And Asc(strS) <= 91 Then ' it's A to Z
        varI2 = intI + 1
        Exit Do
25 End If
    Loop

```

```

    ModelEmbedKey = left(strFN, varI1) & strNewKey & right(strFN, 4)

```

End Function

30 ' returns the key of the next child for this model

```

Private Function NextModelKey(strFN As String) As String

```

```

    Dim nodN As Node
    Dim strNewFN As String
    Dim strIndex As String
35 Dim strT As String

```

```
strIndex = ModelKey(strFN)
```

```
Dim intI As Integer
```

```
' when the key can't be found in the Nodes collection, an error  
' is raised. When the error is raised, the first available letter  
' of the alphabet has been found.
```

```
On Error GoTo Found
```

```
For intI = 65 To 90 ' A thru Z
```

```
    strT = Chr(intI)
```

```
    Set nodN = treModels.Nodes.Item(strIndex & strT)
```

```
Next intI
```

```
On Error GoTo 0
```

```
Call MsgBox("Can't add another child model to this parent", _  
    vbExclamation, "Error")
```

```
Exit Function
```

```
Found:
```

```
NextModelKey = strIndex & strT
```

```
Exit Function
```

```
End Function
```

```
' resets controls and variables when a new family is opened.
```

```
Private Sub ClearControls()
```

```
    If mudtFam Is Nothing Then
```

```
        ' do nothing
```

```
    Else
```

```
        mudtFam.WriteFamily
```

```
        If mudtFam.ActiveModel Is Nothing Then
```

```
            ' do nothing
```

```
        Else
```

```
            mudtFam.ActiveModel.WriteModel
```

```
        End If
```

```
    End If
```

```
mudtWord.CloseAllDocs
```

```
Set mudtFam = Nothing
Set mudtClone = Nothing
```

```
treModels.Nodes.Clear
lstVariables.Clear
5 lstDisposition.Clear
lstAccepted.Clear
stbS.Panels(pnProgramName) = ""
stbS.Panels(pnFamilyName) = ""
10 stbS.Panels(pnItemType) = ""
stbS.Panels(pnGeneric) = ""
stbS.Panels(pnProximity) = ""
stbS.Panels(pnActiveModelIcon).Picture = Nothing
stbS.Panels(pnActiveModelName) = ""
15 frmComments.Comment = ""
mnuAcceptedCopy.Enabled = False
mnuAcceptedPaste.Enabled = False
```

```
End Sub
```

```
' used to reformat tab 2 as QC and DS don't need a distractor listbox
Private Sub FormatTab2(ByVal udtItemType As ItemType)
```

```
20 Select Case udtItemType
    Case ptStandardMC
        ' turn on the distractor list box
        lblDistractor.Visible = True
        lstConstraints(1).Visible = True
        25 cmdConstraintAdd(1).Visible = True
        cmdConstraintEdit(1).Visible = True
        cmdConstraintRemove(1).Visible = True
        cmdConstraintTest(1).Visible = True
    Case ptQuantComp
        30 ' turn off the distractor list box
        lblDistractor.Visible = False
        lstConstraints(1).Visible = False
        cmdConstraintAdd(1).Visible = False
        cmdConstraintEdit(1).Visible = False
        35 cmdConstraintRemove(1).Visible = False
        cmdConstraintTest(1).Visible = False
    Case ptDataSuff
        ' turn off the distractor list box
        lblDistractor.Visible = False
        40 lstConstraints(1).Visible = False
        cmdConstraintAdd(1).Visible = False
```

```
cmdConstraintEdit(1).Visible = False
cmdConstraintRemove(1).Visible = False
cmdConstraintTest(1).Visible = False
End Select
```

```
5 End Sub
```

```
' this method gets rid of all variants in the lstDisposition listbox,
' deletes them from disk, and removes them from the active model.
```

```
Private Sub KillVariants()
```

```
10 Dim udtClone As Clone
Dim intI As Integer

With lstDisposition
    For intI = 0 To .ListCount - 1
        ' get object from active model's clone collection
        Set udtClone = mudtFam.ActiveModel.Clones.Item(Str(.ItemData(intI)))
        ' close the document
        udtClone.CloseDoc
        ' delete the clone file
        Kill IN_DIRECTORY & udtClone.FileName
        ' remove the clone from the active model's collection
        Call mudtFam.ActiveModel.Clones.Remove(Str(.ItemData(intI)))
    Next intI
    For intI = .ListCount - 1 To 0 Step -1
        ' remove the entry from the disposition list box
        Call .RemoveItem(intI)
    Next intI
End With

End Sub
```

```
Private Sub UpdateTab0ControlStates()
```

```
30 ' update model tree menu states
With treModels
    If .Nodes.Count > 0 Then
        mnuTreeExtend.Enabled = True
        mnuTreeRemove.Enabled = True
        cmdTreeExtend.Enabled = True
        cmdTreeRemove.Enabled = True
    Else
        mnuTreeExtend.Enabled = False
    End If
End With
```

```

        mnuTreeRemove.Enabled = False
        cmdTreeExtend.Enabled = False
        cmdTreeRemove.Enabled = False
    End If
5  End With

' update accepted list box menu states
With lstAccepted
    If .ListCount > 0 Then
        cmdPrintBatch.Enabled = True
10    If .SelCount = 1 Then ' 1 item is selected
        mnuAcceptedProfile.Enabled = True
        mnuAcceptedCopy.Enabled = True
        cmdAcceptedEdit.Enabled = True
        cmdAcceptedCopy.Enabled = True
15    ElseIf .SelCount > 1 Then ' more than one is selected
        mnuAcceptedProfile.Enabled = False
        mnuAcceptedCopy.Enabled = False
        cmdAcceptedEdit.Enabled = False
        cmdAcceptedCopy.Enabled = False
20    End If
    Else ' nothings in the list box
        cmdPrintBatch.Enabled = False
        mnuAcceptedProfile.Enabled = False
        mnuAcceptedCopy.Enabled = False
25    mnuAcceptedPaste.Enabled = False
        cmdAcceptedEdit.Enabled = False
        cmdAcceptedCopy.Enabled = False
        cmdAcceptedPaste.Enabled = False
    End If
30 End With

If mudtClone Is Nothing Then ' nothing to paste
    mnuAcceptedPaste.Enabled = False
    cmdAcceptedPaste.Enabled = False
ElseIf lstAccepted.SelCount > 0 Then ' one or more are selected
35    mnuAcceptedPaste.Enabled = True
    cmdAcceptedPaste.Enabled = True
Else ' none are selected
    mnuAcceptedPaste.Enabled = False
    cmdAcceptedPaste.Enabled = False
40 End If

If mudtFam Is Nothing Then
    cmdDone.Enabled = False

```



```
Else
    cmdDone.Enabled = True
End If
```

```
End Sub
```

```
5 Private Sub UpdateTab1ControlStates(Optional ByVal intIndex As Integer = 0)
```

```
    Dim strCaption As String
```

```
    If mudtFam.ActiveModel.IsFrozen Then
        strCaption = "Browse"
```

```
10 Else
        strCaption = "Edit"
    End If
```

```
    mnuVariablesEdit.Caption = strCaption
    cmdVariableEdit.Caption = strCaption
    mnuConstraintsEdit.Caption = strCaption
15 cmdConstraintEdit(0).Caption = strCaption
    cmdConstraintEdit(1).Caption = strCaption
```

```
    ' update variable list box menu states
```

```
20 If mudtFam.ActiveModel.IsFrozen Then
        mnuVariablesAdd.Enabled = False
        mnuVariablesEdit.Enabled = True
        mnuVariablesEnableAll.Enabled = False
        mnuVariablesDisableAll.Enabled = False
        mnuVariablesRemove.Enabled = False
        mnuVariablesRemoveAll.Enabled = False
25 cmdVariableAdd.Enabled = False
        cmdVariableEdit.Enabled = True
        cmdVariableRemove.Enabled = False
```

```
    ElseIf lstVariables.ListCount > 0 Then
        mnuVariablesAdd.Enabled = True
30 mnuVariablesEdit.Enabled = True
        mnuVariablesEnableAll.Enabled = True
        mnuVariablesDisableAll.Enabled = True
        mnuVariablesRemove.Enabled = True
        mnuVariablesRemoveAll.Enabled = True
35 cmdVariableAdd.Enabled = True
        cmdVariableEdit.Enabled = True
        cmdVariableRemove.Enabled = True
```

```
    Else
        mnuVariablesAdd.Enabled = True
```

```

mnuVariablesEdit.Enabled = False
mnuVariablesEnableAll.Enabled = False
mnuVariablesDisableAll.Enabled = False
mnuVariablesRemove.Enabled = False
5 mnuVariablesRemoveAll.Enabled = False
cmdVariableAdd.Enabled = True
cmdVariableEdit.Enabled = False
cmdVariableRemove.Enabled = False
End If

```

```

10 ' isfrozen should not effect state of test option
If lstVariables.ListCount > 0 Then
    mnuVariablesTest.Enabled = True
    cmdVariableTest.Enabled = True
Else
15 mnuVariablesTest.Enabled = False
    cmdVariableTest.Enabled = False
End If

```

```

' update constraints list box menu states
If mudtFam.ActiveModel.IsFrozen Then
20 mnuConstraintsAdd.Enabled = False
    mnuConstraintsEdit.Enabled = True
    mnuConstraintsEnableAll.Enabled = False
    mnuConstraintsDisableAll.Enabled = False
    mnuConstraintsRemove.Enabled = False
25 mnuConstraintsRemoveAll.Enabled = False
    cmdConstraintAdd(0).Enabled = False
    cmdConstraintAdd(1).Enabled = False
    cmdConstraintEdit(0).Enabled = True
    cmdConstraintEdit(1).Enabled = True
30 cmdConstraintRemove(0).Enabled = False
    cmdConstraintRemove(1).Enabled = False
ElseIf lstConstraints(intIndex).ListCount > 0 Then
    mnuConstraintsAdd.Enabled = True
    mnuConstraintsEdit.Enabled = True
35 mnuConstraintsEnableAll.Enabled = True
    mnuConstraintsDisableAll.Enabled = True
    mnuConstraintsRemove.Enabled = True
    mnuConstraintsRemoveAll.Enabled = True
    cmdConstraintAdd(intIndex).Enabled = True
40 cmdConstraintEdit(intIndex).Enabled = True
    cmdConstraintRemove(intIndex).Enabled = True
Else
    mnuConstraintsAdd.Enabled = True

```

```

mnuConstraintsEdit.Enabled = False
mnuConstraintsEnableAll.Enabled = False
mnuConstraintsDisableAll.Enabled = False
mnuConstraintsRemove.Enabled = False
5 mnuConstraintsRemoveAll.Enabled = False
cmdConstraintAdd(intIndex).Enabled = True
cmdConstraintEdit(intIndex).Enabled = False
cmdConstraintRemove(intIndex).Enabled = False
End If

```

```

10 ' isfrozen should not effect state of test option
If lstConstraints(intIndex).ListCount > 0 Then
    mnuConstraintsTest.Enabled = True
    cmdConstraintTest(intIndex).Enabled = True
Else
15     mnuConstraintsTest.Enabled = False
    cmdConstraintTest(intIndex).Enabled = False
End If

```

```

' flip the index
If intIndex = 0 Then
20     intIndex = 1
Else
    intIndex = 0
End If

```

```

' update button states for the other constraint list box
25 If mudtFam.ActiveModel.IsFrozen = False Then
    If lstConstraints(intIndex).ListCount > 0 Then
        cmdConstraintAdd(intIndex).Enabled = True
        cmdConstraintEdit(intIndex).Enabled = True
        cmdConstraintRemove(intIndex).Enabled = True
30     Else
        cmdConstraintAdd(intIndex).Enabled = True
        cmdConstraintEdit(intIndex).Enabled = False
        cmdConstraintRemove(intIndex).Enabled = False
    End If
35 End If

```

```

' isfrozen should not effect state of test option
If lstConstraints(intIndex).ListCount > 0 Then
    cmdConstraintTest(intIndex).Enabled = True
Else
40     cmdConstraintTest(intIndex).Enabled = False
End If

```

```
' update import button
If mudtFam.ActiveModel.IsFrozen Then
    cmdImportConstraints.Enabled = False
Else
5    cmdImportConstraints.Enabled = True
End If
```

```
' if model frozen, disable save
If mudtFam.ActiveModel.IsFrozen Then
    cmdSaveModel.Enabled = False
10 Else
    If mudtFam.ActiveModel.IsDirty Then
        cmdSaveModel.Enabled = True
    Else
        cmdSaveModel.Enabled = False
15 End If
End If
```

```
End Sub
```

```
Private Sub UpdateTab2ControlStates()
```

```
' update disposition list box menu states
20 If lstDisposition.ListCount > 0 And cmdGenerate.Caption = "Generate" Then
    mnuDispAccept.Enabled = True
    mnuDispDefer.Enabled = True
    mnuDispDiscard.Enabled = True
    mnuDispMakeModel.Enabled = True
25 cmdPrintVariants.Enabled = True
    cmdPrintVariants.Enabled = True
    cmdDispAccept.Enabled = True
    cmdDispDefer.Enabled = True
    cmdDispDiscard.Enabled = True
30 cmdDispMakeModel.Enabled = True
Else
    mnuDispAccept.Enabled = False
    mnuDispDefer.Enabled = False
    mnuDispDiscard.Enabled = False
35 mnuDispMakeModel.Enabled = False
    cmdPrintVariants.Enabled = False
    cmdPrintVariants.Enabled = False
    cmdDispAccept.Enabled = False
    cmdDispDefer.Enabled = False
40 cmdDispDiscard.Enabled = False
    cmdDispMakeModel.Enabled = False
```

End If

End Sub

VBSCA -216-

```

' Variable.frm
VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0"; "COMCTL32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
5 Begin VB.Form frmVariable
    BorderStyle   = 4 'Fixed ToolWindow
    Caption       = "Create or Change Variable"
    ClientHeight  = 4230
    ClientLeft    = 45
10    ClientTop    = 285
    ClientWidth   = 6525
    LinkTopic     = "Form1"
    MaxButton     = 0 'False
    MinButton     = 0 'False
15    ScaleHeight  = 4230
    ScaleWidth    = 6525
    ShowInTaskbar = 0 'False
    StartUpPosition = 1 'CenterOwner
    Begin VB.ComboBox cboVarType
20        Height      = 315
        ItemData     = "Variable.frx":0000
        Left         = 2040
        List         = "Variable.frx":0013
        Style        = 2 'Dropdown List
25        TabIndex    = 1
        ToolTipText   = "Select the variable type."
        Top          = 360
        Width        = 1695
    End
30    Begin VB.CheckBox chkChecksum
        Caption      = "Add to checksum"
        Height       = 375
        Left         = 240
        TabIndex     = 2
35        ToolTipText  = "Check this box to add this variable to the checksum calcuation."
        Top          = 840
        Value        = 1 'Checked
        Width        = 1815
    End
40    Begin MSComDlg.CommonDialog cdlCD
        Left        = 5280
        Top         = 2520
        _ExtentX    = 847
        _ExtentY    = 847
    End

```

```

_Version      = 393216
End
Begin VB.CommandButton cmdVarExport
    Caption     = "Export Strings"
    Height      = 495
    Left        = 5160
    TabIndex    = 7
    ToolTipText = "Click here to export a set of strings."
    Top         = 1920
    Width       = 1215
End
Begin VB.CommandButton cmdVarImport
    Caption     = "Import Strings"
    Height      = 495
    Left        = 5160
    TabIndex    = 6
    ToolTipText = "Click here to import a set of strings."
    Top         = 1320
    Width       = 1215
End
Begin VB.TextBox txtVariableName
    Height      = 315
    Left        = 240
    TabIndex    = 0
    ToolTipText = "Enter the variable name here."
    Top         = 360
    Width       = 1695
End
Begin VB.CommandButton cmdVarCancel
    Caption     = "Cancel"
    Height      = 495
    Left        = 5160
    TabIndex    = 5
    ToolTipText = "Click here to return without saving changes."
    Top         = 720
    Width       = 1215
End
Begin VB.CommandButton cmdVarOK
    Caption     = "OK"
    Default     = -1 'True
    Height      = 495
    Left        = 5160
    TabIndex    = 4
    ToolTipText = "Click here to save changes and return."
    Top         = 120

```

Width = 1215
End

Begin ComctlLib.ListView lvwTemp

Height = 375
Left = 5280
TabIndex = 43
Top = 3120
Visible = 0 'False
Width = 495

_ExtentX = 873
_ExtentY = 661
View = 3

Arrange = 2
LabelEdit = 1

MultiSelect = -1 'True
LabelWrap = -1 'True
HideSelection = -1 'True

_Version = 327682
ForeColor = -2147483640
BackColor = -2147483643
BorderStyle = 1
Appearance = 1
NumItems = 0

End

Begin ComctlLib.ListView lvwDummy

Height = 375
Left = 5280
TabIndex = 44
Top = 3600
Visible = 0 'False
Width = 495

_ExtentX = 873
_ExtentY = 661
View = 3

Arrange = 2
LabelEdit = 1

MultiSelect = -1 'True
LabelWrap = -1 'True
HideSelection = -1 'True

_Version = 327682
ForeColor = -2147483640
BackColor = -2147483643
BorderStyle = 1
Appearance = 1
NumItems = 0


```

End
Begin VB.Frame fraString
    BorderStyle  = 0 'None
    Height      = 2895
    Left       = 240
    TabIndex   = 9
    Top        = 1200
    Width      = 4815
    Begin ComctlLib.ListView lvwStrings
        Height   = 1815
        Left     = 0
        TabIndex = 42
        Top      = 720
        Width    = 3975
        _ExtentX = 7011
        _ExtentY = 3201
        View     = 3
        Arrange  = 2
        LabelEdit = 1
        MultiSelect = -1 'True
        LabelWrap = -1 'True
        HideSelection = -1 'True
        _Version  = 327682
        ForeColor = -2147483640
        BackColor = -2147483643
        BorderStyle = 1
        Appearance = 1
        NumItems  = 0
    End
    Begin VB.CheckBox chkIndexed
        Caption   = "Indexed"
        Height    = 375
        Left      = 0
        TabIndex  = 41
        ToolTipText = "Check this box for indexed strings."
        Top       = 0
        Width     = 1215
    End
    Begin VB.CommandButton cmdRemove
        Caption   = "Remove"
        Height    = 255
        Left      = 2640
        TabIndex  = 40
        ToolTipText = "Click here to remove a set of indexed values."
        Top       = 2520
    End

```

```

Width      = 1335
End
Begin VB.CommandButton cmdEdit
Caption     = "Edit"
5   Height   = 255
    Left     = 1320
    TabIndex = 39
    ToolTipText = "Click here to edit a set of indexed values."
10   Top      = 2520
    Width    = 1335
End
Begin VB.CommandButton cmdAdd
Caption     = "Add"
15   Height   = 255
    Left     = 0
    TabIndex = 38
    ToolTipText = "Click here to add a new set of indexed values."
    Top      = 2520
    Width    = 1335
20   End
Begin VB.Label lblStringVals
Caption     = "String values"
25   Height   = 255
    Left     = 0
    TabIndex = 37
    Top      = 480
    Width    = 1695
    End
End
30   Begin VB.Frame fraUntyped
    BorderStyle = 0 'None
    Height      = 2895
    Left        = 240
    TabIndex    = 35
    Top         = 1200
    Width       = 4815
35   Begin VB.TextBox txtUntyped
    Height      = 2295
    Left        = 240
    Locked      = -1 'True
    MultiLine   = -1 'True
    TabIndex    = 36
    ToolTipText = "Interesting, no?"
    Top         = 360
    Width       = 4335
40
45

```

```

End
End
Begin VB.Frame fraIndependent
    BorderStyle  = 0 'None
    Caption      = "Frame1"
    Height       = 2895
    Left         = 240
    TabIndex     = 10
    Top          = 1200
    Width        = 4815
Begin VB.CheckBox chkIsIndependent
    Caption      = "Independent"
    Height       = 375
    Left         = 0
    TabIndex     = 11
    ToolTipText  = "Check this box if the value of this variable is not dependent."
    Top          = 0
    Value        = 1 'Checked
    Width        = 1575
End
Begin VB.Frame fraRealFormat
    BorderStyle  = 0 'None
    Height       = 1095
    Left         = 0
    TabIndex     = 26
    Top          = 1680
    Width        = 4815
Begin VB.CheckBox chkOnGrid
    Caption      = "Value must be multiple of precision"
    Height       = 375
    Left         = 1800
    TabIndex     = 45
    Top          = 120
    Width        = 2895
End
Begin VB.ComboBox cboPrecision
    Height       = 315
    ItemData     = "Variable.frx":0041
    Left         = 120
    List          = "Variable.frx":0060
    Style        = 2 'Dropdown List
    TabIndex     = 34
    Top          = 360
    Width        = 1455
End

```

```

Begin VB.CheckBox chkTrailingZeros
    Caption      = "Display trailing zeros"
    Height       = 375
    Left         = 1800
    TabIndex     = 28
    Top          = 480
    Width        = 1935
End
Begin VB.Label LblDecimals
    Caption      = "Precision"
    Height       = 255
    Left         = 480
    TabIndex     = 29
    Top          = 120
    Width        = 1095
End
Begin VB.Frame fraFractionFormat
    BorderStyle  = 0 'None
    Caption      = "Frame1"
    Height       = 1215
    Left         = -120
    TabIndex     = 32
    Top          = 1560
    Width        = 5055
Begin VB.CheckBox chkMixedNumbers
    Caption      = "Mixed numbers"
    Height       = 375
    Left         = 1560
    TabIndex     = 33
    ToolTipText  = "Check this box if you wish improper fractions to be converted into
mixed numbers."
    Top          = 240
    Width        = 1695
End
End
Begin VB.Frame fraIntRealRange
    BorderStyle  = 0 'None
    Height       = 1335
    Left         = 0
    TabIndex     = 22
    Top          = 360
    Width        = 4815
Begin VB.TextBox txtBy
    Height       = 315

```

```

Left      = 3240
TabIndex  = 25
Text      = "1"
ToolTipText = "Enter the increment here. Variables and expressions may be used."
5 Top      = 600
Width     = 1455
End
Begin VB.TextBox txtTo
10 Height   = 315
Left      = 1680
TabIndex  = 24
Text      = "100"
ToolTipText = "Enter the value in the range here. Variables and expressions may be
used."
15 Top      = 600
Width     = 1455
End
Begin VB.TextBox txtFrom
20 Height   = 315
Left      = 120
TabIndex  = 23
Text      = "1"
ToolTipText = "Enter the lowest value in the range here. Variables and expressions
may be used."
25 Top      = 600
Width     = 1455
End
Begin VB.Label lblBy
30 Caption   = "By"
Height     = 255
Index      = 0
Left       = 3840
TabIndex   = 31
Top        = 360
35 Width    = 495
End
Begin VB.Label lblTo
Caption    = "To"
40 Height  = 255
Index     = 0
Left      = 2280
TabIndex  = 30
Top       = 360
45 Width   = 615
End

```

```

Begin VB.Label lblFrom
    Caption      = "From"
    Height       = 255
    Index        = 0
    Left         = 720
    TabIndex     = 27
    Top          = 360
    Width        = 975
End
End
Begin VB.Frame fraFractionRange
    BorderStyle  = 0 'None
    Height       = 1455
    Left         = 0
    TabIndex     = 12
    Top          = 360
    Width        = 4815
Begin VB.TextBox txtByNum
    Height       = 315
    Left         = 3240
    TabIndex     = 18
    Text         = "1"
    ToolTipText  = "Enter the numerator of the increment here."
    Top          = 360
    Width        = 1455
End
Begin VB.TextBox txtToNum
    Height       = 315
    Left         = 1680
    TabIndex     = 17
    Text         = "100"
    ToolTipText  = "Enter the numerator of the highest value in the range here."
    Top          = 360
    Width        = 1455
End
Begin VB.TextBox txtFromNum
    Height       = 315
    Left         = 120
    TabIndex     = 16
    Text         = "1"
    ToolTipText  = "Enter the numerator of the lowest value of the range here."
    Top          = 360
    Width        = 1455
End
Begin VB.TextBox txtFromDen

```

```

Height      = 315
Left        = 120
TabIndex    = 15
Text        = "1"
5  ToolTipText = "Enter the denominator of the lowest value in the range here."
Top         = 840
Width       = 1455
End
Begin VB.TextBox txtToDen
10  Height      = 315
    Left       = 1680
    TabIndex    = 14
    Text        = "1"
    ToolTipText = "Enter the denominator of the highest value in the range here."
15  Top         = 840
    Width       = 1455
End
Begin VB.TextBox txtByDen
20  Height      = 315
    Left       = 3240
    TabIndex    = 13
    Text        = "1"
    ToolTipText = "Enter the denominator of the increment here."
25  Top         = 840
    Width       = 1455
End
Begin VB.Label lblBy
30  Caption     = "By"
    Height      = 255
    Index       = 1
    Left        = 3840
    TabIndex    = 21
    Top         = 120
    Width       = 255
35  End
Begin VB.Label lblTo
    Caption     = "To"
    Height      = 255
    Index       = 1
40  Left        = 2280
    TabIndex    = 20
    Top         = 120
    Width       = 375
End
45  Begin VB.Label lblFrom

```

```

Caption      = "From"
Height       = 255
Index        = 1
Left         = 480
5  TabIndex   = 19
Top          = 120
Width        = 495
End
Begin VB.Line Line1
10  BorderWidth = 3
    Index      = 0
    X1         = 120
    X2         = 1560
    Y1         = 750
15  Y2         = 750
End
Begin VB.Line Line1
    BorderWidth = 3
    Index       = 1
20  X1         = 1680
    X2         = 3120
    Y1         = 750
    Y2         = 750
End
25  Begin VB.Line Line1
    BorderWidth = 3
    Index       = 2
    X1         = 3240
    X2         = 4680
30  Y1         = 750
    Y2         = 750
End
End
End
35  Begin VB.Label lblVarType
    Caption     = "Type"
    Height      = 255
    Left        = 2040
    TabIndex    = 8
40  Top        = 120
    Width       = 1095
End
Begin VB.Label lblVarName
45  Caption     = "Variable Name"
    Height      = 255

```



```

        Left      = 240
        TabIndex  = 3
        Top       = 120
        Width     = 1095
5      End
      Begin VB.Menu mnuString
        Caption    = "String"
        Visible    = 0 'False
      Begin VB.Menu mnuStringAdd
10      Caption    = "Add"
      End
      Begin VB.Menu mnuStringEdit
        Caption    = "Edit"
      End
15      Begin VB.Menu mnuStringRemove
        Caption    = "Remove"
      End
    End
  End
20  Attribute VB_Name = "frmVariable"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = True
  Attribute VB_Exposed = False
25  Option Explicit

  Private mudtVar As Variable
  Private mudtVarInt As VarInteger
  Private mudtVarReal As VarReal
  Private mudtVarFraction As VarFraction
30  Private mudtVarString As VarString
  Private mudtVarUntyped As VarUntyped

  ' to see if the variable type has changed
  Private mudtType As VariableType
  Private mudtOldType As VariableType

35  ' needed for string list box
  Private mbytAddEditFlag As Byte

  ' needed for listbox update
  Private mlstListBox As ListBox

  'current active model
40  Private mudtModel As Model

```

```
Public Property Let AddEditFlag(ByVal bytNewValue As Byte)
```

```
    mbytAddEditFlag = bytNewValue
```

```
End Property
```

```
Public Property Get AddEditFlag() As Byte
```

```
5    AddEditFlag = mbytAddEditFlag
```

```
End Property
```

```
Public Property Let Variable(ByVal udtNewValue As Variable)
```

```
    Set mudtVar = udtNewValue
```

```
10 End Property
```

```
Public Property Let ListBox(ByVal lstNewValue As ListBox)
```

```
    Set mlstListBox = lstNewValue
```

```
End Property
```

```
Public Property Let Model(ByVal udtNewValue As Model)
```

```
15 Set mudtModel = udtNewValue
```

```
End Property
```

```
Private Sub chkIndexed__Click()
```

```
    Call CopyListView(lvwStrings, lvwTemp)
```

```
    Call CopyListView(lvwDummy, lvwStrings)
```

```
20    Call CopyListView(lvwTemp, lvwDummy)
```

```
End Sub
```

```
Private Sub CopyListView(ByVal lvw1 As ListView, lvw2 As ListView)
```

```
    Dim intI As Integer
```

```
    Dim intI2 As Integer
```

```
25    Dim lsiItem As ListItem
```

```
    ' copy visible listview into temp listview
```

```
lvw2.ListItems.Clear
lvw2.ColumnHeaders.Clear
```

```
For intI = 1 To lvw1.ColumnHeaders.Count
5   Call lvw2.ColumnHeaders.Add(, , lvw1.ColumnHeaders(intI))
Next intI
```

```
For intI = 1 To lvw1.ListItems.Count
10   Set lsiItem = lvw2.ListItems.Add(, , lvw1.ListItems.Item(intI).Text)
   For intI2 = 1 To lvw1.ColumnHeaders.Count - 1
       lsiItem.SubItems(intI2) = lvw1.ListItems.Item(intI).SubItems(intI2)
   Next intI2
Next intI
```

```
15 End Sub
```

```
Private Sub cmdAdd_Click()
```

```
    Call mnuStringAdd_Click
```

```
End Sub
```

```
20 Private Sub cmdEdit_Click()
```

```
    Call mnuStringEdit_Click
```

```
End Sub
```

```
Private Sub cmdRemove_Click()
```

```
    Call mnuStringRemove_Click
```

```
25 End Sub
```

```
Private Sub Form_Load()
```

```
    Dim udtWAPI As New Win32API
```

```
    ' enable full row select
```

```
30   Call udtWAPI.EnableListViewFullRowSelect(lvwStrings)
```

```
    ' load up explanation of untyped variables
```

```
    txtUntyped = LoadResString(1)
```

```
    ' cboVarDelimiter.ListIndex = 0 ' default to "@"
```

cboPrecision.ListIndex = 1 ' default to ".01"

cdlCD.CancelError = True

If mbytAddEditFlag = aeEdit Then

txtVariableName = mudtVar.name

If mudtVar.Checksum Then

chkChecksum = 1

Else

chkChecksum = 0

End If

Select Case TypeName(mudtVar)

Case "VarInteger"

Set mudtVarInt = mudtVar

With mudtVarInt

txtFrom = .From

txtTo = .Too

txtBy = .By

If .IsIndependent Then

chkIsIndependent = 1

Else

chkIsIndependent = 0

End If

End With

mudtType = vtInteger

Case "VarReal"

Set mudtVarReal = mudtVar

With mudtVarReal

txtFrom = .From

txtTo = .Too

txtBy = .By

If .IsIndependent Then

chkIsIndependent = 1

Else

chkIsIndependent = 0

End If

If .IsOnGrid Then

chkOnGrid = 1

Else

chkOnGrid = 0

```
End If
If .TrailingZeros Then
    chkTrailingZeros = 1
Else
    chkTrailingZeros = 0
End If
cboPrecision = .Precision
End With
mudtType = vtReal
```

Case "VarFraction"

```
Set mudtVarFraction = mudtVar
With mudtVarFraction
    txtFromNum = .FromNumerator
    txtFromDen = .FromDenominator
    txtToNum = .ToNumerator
    txtToDen = .ToDenominator
    txtByNum = .ByNumerator
    txtByDen = .ByDenominator
    If .IsIndependent Then
        chkIsIndependent = 1
    Else
        chkIsIndependent = 0
    End If
    If .MixedNumbers Then
        chkMixedNumbers = 1
    Else
        chkMixedNumbers = 0
    End If
End With
mudtType = vtFraction
```

Case "VarString"

```
Set mudtVarString = mudtVar
With mudtVarString
    mudtType = vtString
    If .Delimiter = Chr(STRING_DELIMITER) Then
        ' do nothing
    Else
        ConvertDelimiter
        .Delimiter = Chr(STRING_DELIMITER)
    End If
    ' load list view control
    If .IsIndexed Then
        chkIndexed = 1
```

```

        Else
            chkIndexed = 0
        End If
        LoadListView
5      End With

        Case "VarUntyped"
            Set mudtVarUntyped = mudtVar
            mudtType = vtUntyped
10      End Select

        mudtOldType = mudtType
        cboVarType.ListIndex = mudtType 'generates a cboVarType_Click event
15      Else ' it's an add

            mudtType = vtInteger
            mudtOldType = mudtType
            cboVarType.ListIndex = vtInteger 'generates a cboVarType_Click event
20      End If

        ' changes control states if model is frozen
25      UpdateControlStates

    End Sub

    Private Sub cmdVarOK_Click()

        ' will capitalize the first letter of the variable name, if it's not
        ' capitalized already.
30      txtVariableName_LostFocus

        ' make sure all input is valid, otherwise, make 'em fix it!
        If ValidateForm = False Then
            Exit Sub
35      End If

        If mbytAddEditFlag = aeEdit Then ' we're editing an old one
            Call ProcessEdit
        Else
40      Call ProcessAdd
        End If

```

Unload Me

End Sub

5 Private Sub cmdVarCancel_Click()

Unload Me

End Sub

Private Sub cmdVarImport_Click()

10

Dim strFN As String

With cdlCD

.FileName = ""

15

.DialogTitle = "Import strings from file"

.Filter = "String Files (*.str)|*.str|"

.DefaultExt = ".str"

.InitDir = "c:\tcs\tca\strings"

.Flags = cdlOFNFileMustExist + cdlOFNHideReadOnly

20

On Error GoTo Cancel

.ShowOpen

On Error GoTo 0

strFN = .FileName

End With

25

On Error GoTo BeatIt ' trap open, I/O errors

Open strFN For Input Access Read As 1

30

Dim varR As Variant

Dim varIndexed As Variant

Dim varNumIndices As Variant

Dim strMessage As String

Dim mcolStr As Collection

35

Dim intI As Integer

Input #1, varIndexed

If varIndexed Then

40

strMessage = "indexed."

Else

strMessage = "not indexed."

End If

If varIndexed <> chkIndexed Then

Call MsgBox("Unable to import: file contains string values that are " & _
strMessage, vbExclamation, "Error")

GoTo BeatIt

End If

Input #1, varNumIndices

Do

Input #1, varR

If varIndexed Then

Set mcolStr = New Collection

Call mcolStr.Add(varR)

For intI = 1 To varNumIndices - 1

Input #1, varR

Call mcolStr.Add(varR)

Next intI

Call AddColToListView(mcolStr)

Else

Call AddStrToListView(varR)

End If

Loop Until EOF(1)

BeatIt:

Close 1

Cancel:

Exit Sub

End Sub

Private Sub cmdVarExport_Click()

Dim strFN As String

cdlCD.CancelError = True

With cdlCD

.FileName = ""

.DialogTitle = "Export strings to file"

.Filter = "String Files (*.str)|*.str|"

.DefaultExt = ".txt"


```

        .InitDir = "c:\tcs\tca\strings"
        .Flags = cdlOFNOverwritePrompt + cdlOFNHideReadOnly
        On Error GoTo Cancel
        .ShowSave
5       On Error GoTo 0
        strFN = .FileName
End With

On Error GoTo BeatIt

10      Open strFN For Output Access Write As 1

        Dim varW As Variant

15      varW = chkIndexed ' so we can tell if it's indexed
        Print #1, varW
        varW = lvwStrings.ColumnHeaders.Count ' how many indices
        Print #1, varW

20      Dim intI As Integer
        Dim intI2 As Integer
        Dim lsiItem As ListItem

        intI = 1

25      Do ' write the data
        Set lsiItem = lvwStrings.ListItems.Item(intI)
        varW = lsiItem.Text
        Print #1, varW

30      If chkIndexed Then
        For intI2 = 1 To lvwStrings.ColumnHeaders.Count - 1
            varW = lsiItem.SubItems(intI2)
            Print #1, varW
35      Next intI2
        End If

        intI = intI + 1

40      Loop Until intI > lvwStrings.ListItems.Count

BeatIt:
    Close 1

Cancel:

```

Exit Sub

End Sub

Private Sub lvwStrings_MouseDown(Button As Integer, Shift As Integer, _
5 X As Single, Y As Single)

If Button = vbRightButton Then
 PopupMenu mnuString
End If

10 End Sub

Private Sub mnuStringAdd_Click()

If chkIndexed Then
 With frmIndexedString
 ' set the model
15 .Model = mudtModel
 ' set the edit flag
 .AddEditFlag = aeAdd
 ' set var name
 .VariableName = txtVariableName
20 ' do it
 .Show vbModal
 If .OK Then
 Call AddColToListView(.SubStringCollection)
 End If
25 End With
Else

With frmString
 ' set the model
30 .Model = mudtModel
 ' set the string
 .StringValue = ""
 ' set var name
 .VariableName = txtVariableName
 ' do it
35 .Show vbModal
 If .OK Then
 Call AddStrToListView(.StringValue)
 End If
 End With
40 End If

UpdateControlStates

End Sub

Private Sub mnuStringEdit_Click()

5 Dim colC As Collection

If lvwStrings.SelectedItem Is Nothing Then Exit Sub ' Make sure list item is selected

If chkIndexed Then

10 With frmIndexedString

' set the model

.Model = mudtModel

' set the edit flag

.AddEditFlag = aeEdit

15 ' set the substring collection

.SubStringCollection = GetSubStringCollection(lvwStrings.SelectedItem)

' set var name

.VariableName = txtVariableName

' do it

20 .Show vbModal

If .OK Then

Call UpdateListView(lvwStrings.SelectedItem, .SubStringCollection)

End If

End With

25 Else

With frmString

' set the model

.Model = mudtModel

' set the string

30 .StringValue = lvwStrings.SelectedItem

' set var name

.VariableName = txtVariableName

' do it

.Show vbModal

35 If .OK Then

Set colC = New Collection

Call colC.Add(.StringValue)

Call UpdateListView(lvwStrings.SelectedItem, colC)

End If

40 End With

End If

End Sub

```
Private Sub mnuStringRemove_Click()
```

```
    If lvwStrings.SelectedItem Is Nothing Then Exit Sub
```

```
    If MsgBox("Remove string value " & lvwStrings.SelectedItem.Text & "?", _  
        vbQuestion + vbYesNo) = vbNo Then
```

```
        Exit Sub
```

```
    End If
```

```
    With lvwStrings
```

```
        Call .ListItems.Remove(.SelectedItem.index)
```

```
    End With
```

```
    UpdateControlStates
```

```
End Sub
```

```
Private Sub chkIsIndependent_Click()
```

```
    Call FormatForm
```

```
End Sub
```

```
Private Sub cboVarType_Click()
```

```
    mudtType = cboVarType.ListIndex
```

```
    Call FormatForm
```

```
End Sub
```

```
Private Sub txtVariableName_GotFocus()
```

```
    ' Automatically select all text when TextBox gets focus
```

```
    Call txtSelectAll(txtVariableName)
```

```
End Sub
```

```
Private Sub txtVariableName_LostFocus()
```

```
    Dim strName As String
```

```
    Dim udtVar As Variable
```

```
    ' Capitalize the variable name in the textbox
```

```
    strName = txtVariableName
```

```
Call CapitalizeString(strName)
txtVariableName = strName
```

```
End Sub
```

```
5 Private Sub txtFrom_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtFrom)
```

```
End Sub
```

```
10 Private Sub txtTo_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtTo)
```

```
End Sub
```

```
15 Private Sub txtBy_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtBy)
```

```
End Sub
```

```
20 Private Sub txtFromNum_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtFromNum)
```

```
End Sub
```

```
25 Private Sub txtFromDen_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtFromDen)
```

```
End Sub
```

```
30 Private Sub txtToNum_GotFocus()
```

```
' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtToNum)
```

End Sub

Private Sub txtToDen_GotFocus()

' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtToDen)

End Sub

Private Sub txtByNum_GotFocus()

' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtByNum)

End Sub

Private Sub txtByDen_GotFocus()

' Automatically select all text when TextBox gets focus
Call txtSelectAll(txtByDen)

End Sub

Private Sub FormatForm()

cmdVarImport.Visible = False
cmdVarExport.Visible = False

chkIsIndependent.TabStop = False
txtFrom.TabStop = False
txtTo.TabStop = False
txtBy.TabStop = False
txtFromNum.TabStop = False
txtFromDen.TabStop = False
txtToNum.TabStop = False
txtToDen.TabStop = False
txtByNum.TabStop = False
txtByDen.TabStop = False
lvwStrings.TabStop = False
chkTrailingZeros.TabStop = False
chkTrailingZeros.TabStop = False
chkMixedNumbers.TabStop = False

Select Case mudtType

```

Case vtInteger
    fraFractionRange.Visible = False
    fraFractionFormat.Visible = False
    fraIndependent.ZOrder
5    fraIntRealRange.ZOrder
    fraRealFormat.Visible = False
    chkIsIndependent.TabStop = True
    If chkIsIndependent Then
        fraIntRealRange.Visible = True
10        txtFrom.TabStop = True
        txtTo.TabStop = True
        txtBy.TabStop = True
    Else
        fraIntRealRange.Visible = False
15    End If

```

```

Case vtReal
    fraFractionRange.Visible = False
    fraFractionFormat.Visible = False
    fraIndependent.ZOrder
20    fraIntRealRange.ZOrder
    fraRealFormat.ZOrder
    fraRealFormat.Visible = True
    chkIsIndependent.TabStop = True
    If chkIsIndependent Then
        fraIntRealRange.Visible = True
        txtFrom.TabStop = True
        txtTo.TabStop = True
        txtBy.TabStop = True
    Else
        fraIntRealRange.Visible = False
    End If
    chkOnGrid.TabStop = True
    chkTrailingZeros.TabStop = True
35

```

```

Case vtFraction
    fraIntRealRange.Visible = False
    fraRealFormat.Visible = False
    fraIndependent.ZOrder
40    fraFractionRange.ZOrder
    fraFractionFormat.ZOrder
    fraFractionFormat.Visible = True
    chkIsIndependent.TabStop = True
    If chkIsIndependent Then
        fraFractionRange.Visible = True
45

```

```

        txtFromNum.TabStop = True
        txtFromDen.TabStop = True
        txtToNum.TabStop = True
        txtToDen.TabStop = True
5       txtByNum.TabStop = True
        txtByDen.TabStop = True
    Else
        fraFractionRange.Visible = False
    End If
10     chkMixedNumbers.TabStop = True

    Case vtString
        fraString.ZOrder
        cmdVarImport.Visible = True
15     cmdVarExport.Visible = True

    Case vtUntyped
        fraUntyped.ZOrder

20 End Select

    Dim intTabIndex As Integer

    intTabIndex = 4

25     Call AddTab(chkIsIndependent, intTabIndex)
    Call AddTab(txtFrom, intTabIndex)
    Call AddTab(txtTo, intTabIndex)
    Call AddTab(txtBy, intTabIndex)
30     Call AddTab(txtFromNum, intTabIndex)
    Call AddTab(txtFromDen, intTabIndex)
    Call AddTab(txtToNum, intTabIndex)
    Call AddTab(txtToDen, intTabIndex)
    Call AddTab(txtByNum, intTabIndex)
35     Call AddTab(txtByDen, intTabIndex)
    Call AddTab(chkTrailingZeros, intTabIndex)
    Call AddTab(chkOnGrid, intTabIndex)
    Call AddTab(chkMixedNumbers, intTabIndex)

End Sub

40 ' add a tab, if its active
Private Sub AddTab(ByVal ctlC As Control, intIndex As Integer)

    If ctlC.TabStop Then

```



```

        ctlC.TabIndex = intIndex
        intIndex = intIndex + 1
    End If

```

```
End Sub
```

```
5 Private Function ValidateForm() As Boolean
```

```
    ValidateForm = False
```

```
    ' check variable name length > 0
```

```
    If Len(txtVariableName) = 0 Then
```

```
10        Call MsgBox("Variable names must be 1 or more characters long.", _
            vbExclamation, "Error")
        txtVariableName.SetFocus
        Exit Function
    End If
```

```
15    'check first character for alpha
    If Asc(txtVariableName) < 65 Or Asc(txtVariableName) > 91 Then
        Call MsgBox("Variable names must begin in a letter", _
            vbExclamation, "Error")
20        txtVariableName.SetFocus
        Exit Function
    End If
```

```
25    ' check for unique variable name
    Dim blnUnique As Boolean
    blnUnique = True
```

```
    Select Case mbytAddEditFlag
```

```
        Case aeAdd
            blnUnique = mudtModel.Variables.UniqueName(txtVariableName)
```

```
30        Case aeEdit
            blnUnique = mudtModel.Variables.UniqueName(txtVariableName, 1, mudtVar)
```

```
    End Select
```

```
    If blnUnique = False Then
```

```
35        Call MsgBox("Variable name is already in use.", vbExclamation, "Error")
        txtVariableName.SetFocus
        Exit Function
    End If
```

```

' if integer or real, validate contents of From, To, By
If cboVarType = "Integer" Or cboVarType = "Real" Then
    If Not ValidateRange Then
5         Call MsgBox("Entries in From, To, and By must be either a number " & _
            "or a string variable containing a numeric value. " & _
            "Expressions or math variables are not permitted.", _
            vbExclamation, "Error")
        Exit Function
10    End If
End If

```

```

ValidateForm = True

```

```

End Function

```

```

15 Private Function ValidateRange() As Boolean

```

```

    Dim conC As Control
    Dim colC As New Collection
    Dim udtV As Variable
    Dim udtVS As VarString
20    Dim intI As Integer
    Dim blnOK As Boolean

```

```

    Call colC.Add(txtFrom)
    Call colC.Add(txtTo)
25    Call colC.Add(txtBy)

```

```

    For Each conC In colC
        blnOK = False
        If IsNumeric(conC) Then
30            blnOK = True
        Else ' see if the box contains a string variable
            For Each udtV In mudtModel.Variables
                If udtV.Typ = vtString Then
                    Set udtVS = udtV
35                    If udtVS.IsIndexed Then
                        For intI = 1 To udtVS.NumIndices
                            If conC = GetIndexedName(udtV.name, intI) Then
                                blnOK = True
                                Exit For
40                            End If
                        Next intI
                    ElseIf conC = udtV.name Then

```

```

        blnOK = True
    End If
End If
If blnOK Then
    Exit For
End If
Next udtV
End If
If Not blnOK Then
    ValidateRange = False
    Exit Function
End If
Next conC

```

```

ValidateRange = True

```

```

End Function

```

```

Private Sub ProcessEdit()

```

```

    ' Check to see if the type has changed
    If mudtType <> mudtOldType Then

```

```

        With mlstListBox

```

```

            ' remove the old variable from the collection
            Call mudtModel.Variables.Remove(Str(.ItemData(.ListIndex)))
            ' add the new variable
            Call AddVariable
            ' update the index in the list box
            .ItemData(.ListIndex) = mudtVar.index
            ' replace the text in the list box
            .List(.ListIndex) = mudtVar.ScreenFormat
        End With

```

```

    Else

```

```

        ' update it with new data from form
        Select Case mudtType

```

```

            Case vtInteger

```

```

                Call mudtVarInt.Update(txtVariableName, _
                    txtFrom, txtTo, txtBy, _
                    chkIsIndependent, chkChecksum)

```

```

            Case vtReal

```

```

                Call mudtVarReal.Update(txtVariableName, _
                    txtFrom, txtTo, txtBy, chkIsIndependent, _

```

chkChecksum, chkTrailingZeros.Value, cboPrecision, chkOnGrid)

Case vtFraction

Call mudtVarFraction.Update(txtVariableName, _
txtFromNum, txtFromDen, txtToNum, txtToDen, _
txtByNum, txtByDen, chkIsIndependent, chkChecksum, _
chkMixedNumbers)

Case vtString

Dim intI As Integer
Dim intI2 As Integer
Dim colStr As Collection
Dim udtSS As SubString

mudtVar.name = txtVariableName
mudtVar.Checksum = chkChecksum
mudtVarString.IsIndexed = chkIndexed

' build a new collection of strings

Set colStr = New Collection

With lvwStrings

For intI = 1 To (.ListItems.Count)

Set udtSS = New SubString

udtSS.Delimiter = mudtVarString.Delimiter

Call udtSS.AddSubString(.ListItems.Item(intI).Text)

For intI2 = 1 To .ColumnHeaders.Count - 1

Call udtSS.AddSubString(.ListItems.Item(intI).SubItems(intI2))

Next intI2

Call colStr.Add(udtSS.StringValue)

Next intI

End With

mudtVarString.StringCollection = colStr

End Select

With mlstListBox

' replace the text in the list box

.List(.ListIndex) = mudtVar.ScreenFormat

End With

End If

End Sub

Private Sub ProcessAdd()

Call AddVariable

With mlstListBox

' Add the new variable to the variable list box

Call .AddItem(mudtVar.ScreenFormat)

' Set ItemData to index value of the variable object

.ItemData(.ListCount - 1) = mudtVar.index

' Check the check box

.Selected(.ListCount - 1) = True

End With

End Sub

Private Sub AddVariable()

' Add the new variable

Select Case mudtType

Case vtInteger

Set mudtVar = mudtModel.Variables.AddInteger(txtVariableName, _
True, txtFrom, txtTo, txtBy, chkIsIndependent, _
chkChecksum)

Case vtReal

Set mudtVar = mudtModel.Variables.AddReal(txtVariableName, _
True, txtFrom, txtTo, txtBy, chkIsIndependent, _
chkChecksum, chkTrailingZeros.Value, cboPrecision, chkOnGrid)

Case vtFraction

Set mudtVar = mudtModel.Variables.AddFraction(txtVariableName, _
True, txtFromNum, txtFromDen, txtToNum, txtToDen, _
txtByNum, txtByDen, chkIsIndependent, chkChecksum, _
chkMixedNumbers)

Case vtString

Dim intI As Integer

Dim intI2 As Integer

Dim colStr As New Collection

Dim udtSS As SubString

With lvwStrings

For intI = 1 To (.ListItems.Count)

Set udtSS = New SubString

udtSS.Delimiter = Chr(STRING_DELIMITER)

udtSS.AddSubString (.ListItems.Item(intI).Text)

```

        For intI2 = 1 To .ColumnHeaders.Count - 1
            Call udtSS.AddSubString(.ListItems.Item(intI).SubItems(intI2))
        Next intI2
        Call colStr.Add(udtSS.StringValue)
5       Next intI
    End With
    Set mudtVar = mudtModel.Variables.AddString(txtVariableName, True, _
        chkChecksum, Chr(STRING_DELIMITER), chkIndexed, colStr)

10     Case vtUntyped
        Set mudtVar = mudtModel.Variables.AddUntyped(txtVariableName, True, _
            chkChecksum)

    End Select

15 End Sub

Private Sub UpdateControlStates()

    Dim conC As Control

    On Error Resume Next

20     ' shut off all controls that have an enabled property
    For Each conC In Me
        If mudtModel.IsFrozen Then
            conC.Enabled = False
        Else
25         conC.Enabled = True
        End If
    Next conC

    On Error GoTo 0

    ' these stay on even if model is frozen
30     cmdVarCancel.Enabled = True
    fraString.Enabled = True
    lvwStrings.Enabled = True
    cmdEdit.Enabled = True
    mnuStringEdit.Enabled = True

35     ' if model is frozen, change caption of edit button, menu to browse
    If mudtModel.IsFrozen Then
        cmdEdit.Caption = "Browse"
        mnuStringEdit.Caption = "Browse"
    End If
End Sub

```

End If

' turn export on if there's something to export
cmdVarExport.Enabled = CBool(lvwStrings.ListItems.Count)

5 ' shut off "edit", "remove" buttons, menus if the listview is empty
If lvwStrings.ListItems.Count = 0 Then
 mnuStringEdit.Enabled = False
 cmdEdit.Enabled = False
 mnuStringRemove.Enabled = False
10 cmdRemove.Enabled = False
End If

End Sub

' this is used to convert version 0.6 indexed strings to version 0.7 style

15 Private Sub ConvertDelimiter()

Dim colStr As Collection
Dim varS As Variant

With mudtVarString

20 Set colStr = .StringCollection
For Each varS In colStr
 varS = ReplaceAll(varS, .Delimiter, Chr(STRING_DELIMITER))
Next varS
End With

25 End Sub

Private Sub LoadListView()

Dim intI As Integer
Dim varS As Variant

30

With mudtVarString

If chkIndexed Then

' build column headers

For intI = 1 To .NumIndices - 1

35 Call lvwStrings.ColumnHeader.Add(, _
Str(intI), lvwStrings.Width / 4)

Next intI

End If

' fill in values

```

        For Each varS In .StringCollection
            Call AddStrToListView(varS)
        Next varS
    End With

```

5 End Sub

```

Private Sub AddColToListView(ByVal colS As Collection)

```

```

    Dim lsiLI As ListItem

```

```

    Set lsiLI = lvwStrings.ListItems.Add(, , "")

```

10 Call UpdateListView(lsiLI, colS)

```

End Sub

```

```

Private Sub AddStrToListView(ByVal strS As String)

```

```

    Dim udtSS As New SubString

```

```

    Dim lsiLI As ListItem

```

```

    Dim intI As Integer

```

```

    Set lsiLI = lvwStrings.ListItems.Add(, , "")

```

```

    udtSS.Delimiter = Chr(STRING_DELIMITER)

```

```

    udtSS.StringValue = strS

```

```

    Call UpdateListView(lsiLI, udtSS.StringCollection)

```

```

End Sub

```

```

Private Sub UpdateListView(ByVal lsiLI As ListItem, ByVal colS As Collection)

```

```

    Dim intI As Integer

```

```

    Dim intW As Integer

```

```

    Dim strColHeading As String

```

```

    If chkIndexed Then

```

```

        intW = 4

```

```

    Else

```

```

        intW = 1

```

```

    End If

```

```

    ' expand the number of columns if there aren't enough

```

```

    For intI = lvwStrings.ColumnHeaders.Count To colS.Count - 1

```

```

        If chkIndexed Then

```

```

            strColHeading = Str(intI + 1)

```



```
        Call lvwStrings.ColumnHeaders.Add( , strColHeading, _  
            lvwStrings.Width / intW)
```

```
    Else
```

```
        strColHeading = " "
```

```
5        Call lvwStrings.ColumnHeaders.Add( , strColHeading)
```

```
    End If
```

```
Next intI
```

```
' plug in the values
```

```
10 For intI = 1 To colS.Count
```

```
    If intI = 1 Then
```

```
        lsiLI = colS.Item(intI)
```

```
    Else
```

```
        lsiLI.SubItems(intI - 1) = colS.Item(intI)
```

```
15    End If
```

```
Next intI
```

```
' get rid of anything in the list view past colS.Count
```

```
For intI = colS.Count + 1 To lvwStrings.ColumnHeaders.Count
```

```
20 If intI > 1 Then
```

```
    lsiLI.SubItems(intI - 1) = ""
```

```
    Else
```

```
        lsiLI = ""
```

```
    End If
```

```
25 Next intI
```

```
Dim blnEmpty As Boolean
```

```
' get rid of columns with all "" from right to left
```

```
30 ' stop when first column with any string > 0 length is encountered
```

```
For intI = lvwStrings.ColumnHeaders.Count To 1 Step -1
```

```
    For Each lsiLI In lvwStrings.ListItems
```

```
        blnEmpty = True
```

```
        If intI > 1 Then
```

```
35            If lsiLI.SubItems(intI - 1) <> "" Then
```

```
                blnEmpty = False
```

```
                Exit For
```

```
            End If
```

```
        ElseIf lsiLI <> "" Then
```

```
40            blnEmpty = False
```

```
            Exit For
```

```
        End If
```

```
    Next lsiLI
```

```
    If blnEmpty Then
```

```
45        Call lvwStrings.ColumnHeaders.Remove(intI)
```

```
Else
    Exit For
End If
Next intI
```

```
Dim intI2 As Integer
```

```
' get rid of rows with "" in all columns from the bottom up
```

```
For intI2 = lvwStrings.ListItems.Count To 1 Step -1
```

```
    Set lsiLI = lvwStrings.ListItems.Item(intI2)
```

```
    For intI = 1 To lvwStrings.ColumnHeaders.Count
```

```
        blnEmpty = True
```

```
        If intI > 1 Then
```

```
            If lsiLI.SubItems(intI - 1) <> "" Then
```

```
                blnEmpty = False
```

```
                Exit For
```

```
            End If
```

```
        ElseIf lsiLI <> "" Then
```

```
            blnEmpty = False
```

```
            Exit For
```

```
        End If
```

```
    Next intI
```

```
    If blnEmpty Then
```

```
        Call lvwStrings.ListItems.Remove(intI2)
```

```
    End If
```

```
Next intI2
```

```
End Sub
```

```
Private Function GetSubStringCollection(ByVal lsiLI As ListItem) As Collection
```

```
    Dim colC As New Collection
```

```
    Dim intI As Integer
```

```
    Call colC.Add(lsiLI)
```

```
    For intI = 1 To lvwStrings.ColumnHeaders.Count - 1
```

```
        Call colC.Add(lsiLI.SubItems(intI))
```

```
    Next intI
```

```
    Set GetSubStringCollection = colC
```

```
End Function
```

```

' Application.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "TCAApplication"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
    Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
    Option Explicit

    Public Sub Run()

        ' Dim udtP As New Prolog
20        '
        ' If udtP.StartProlog("hlp4lib.p4") = False Then
        '     Call MsgBox("Prolog failure on startup", vbExclamation, "Error")
        ' End If
        '
25        frmTCA.Show

    End Sub

```

```

' CClones.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1 'True
5  END
  Attribute VB_Name = "CClones"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = True
  Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
  Option Explicit

  ' enable i/o
  Private mudtFile As File

  'to hold collection
15  Private mcolClones As Collection

  ' the sequence number appended to clone filenames
  Private mintSeqNum As Integer

  ' is dirty
  Private mblnIsDirty As Boolean

20  Private Sub Class_Initialize()

    'creates the collection when this class is created
    Set mcolClones = New Collection

  End Sub

25  Private Sub Class_Terminate()

    'destroys collection when this class is terminated
    Set mcolClones = Nothing

  End Sub

30  Public Property Get Item(vntIndexKey As Variant) As Clone

    'used when referencing an element in the collection
    'vntIndexKey contains either the Index or Key to the collection,
    'this is why it is declared as a Variant
    'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)

```

```
Set Item = mcolClones(vntIndexKey)
```

```
End Property
```

```
Public Property Get Count() As Long
```

```
5      'used when retrieving the number of elements in the  
      'collection. Syntax: Debug.Print x.Count  
      Count = mcolClones.Count
```

```
End Property
```

10

```
Public Property Get NextSeqNum() As Integer
```

```
      mintSeqNum = mintSeqNum + 1  
      NextSeqNum = mintSeqNum
```

15

```
      mblnIsDirty = True
```

```
End Property
```

```
Public Property Let SeqNum(ByVal intNewValue As Integer)
```

```
      mintSeqNum = intNewValue
```

20

```
      mblnIsDirty = True
```

```
End Property
```

```
Public Property Get SeqNum() As Integer
```

```
      SeqNum = mintSeqNum
```

25

```
End Property
```

```
Public Property Get IsDirty() As Boolean
```

```
      Dim udtClone As Clone
```

30

```
      ' see if any collection members are dirty
```

```
      If Not mblnIsDirty Then
```

```
          For Each udtClone In mcolClones
```

```
              If udtClone.IsDirty Then
```

```
                  mblnIsDirty = True
```

35

```
                  Exit For
```

```
End If
Next udtClone
End If
```

```
5 IsDirty = mblnIsDirty
```

```
End Property
```

```
Private Function NextID() As Long
```

```
' creates a unique index to associate a clone and a listbox
```

```
10 Static lngID As Long
```

```
lngID = lngID + 1
```

```
NextID = lngID
```

```
15 End Function
```

```
Public Function Add(ByVal strFN As String, _  
Optional ByVal blnAddSeqNum = False) As Clone
```

```
Dim udtClone As New Clone
```

```
20 ' add the clone sequence number to the file name if blnAddSeqNum is True.
```

```
If blnAddSeqNum Then
```

```
udtClone.FileName = left(strFN, Len(strFN) - 4) & _  
Trim(Str(NextSeqNum)) & ".doc"
```

```
Else
```

```
25 udtClone.FileName = ExtractFileName(strFN)
```

```
End If
```

```
udtClone.Index = NextID
```

```
30 ' use index of the clone as the key
```

```
Call mcolClones.Add(udtClone, Str(udtClone.Index))
```

```
Set Add = udtClone
```

```
End Function
```

```
35 Public Function AddObj(ByVal udtClone As Clone) As Clone
```

```
udtClone.Index = NextID
```

```
' use index of the clone as the key
```

Call mcolClones.Add(udtClone, Str(udtClone.Index))

Set AddObj = udtClone

End Function

5 Public Sub Remove(vntIndexKey As Variant)

'used when removing an element from the collection

'vntIndexKey contains either the Index or Key, which is why

'it is declared as a Variant

'Syntax: x.Remove(xyz)

10 mcolClones.Remove vntIndexKey

mblnIsDirty = True

End Sub

15 Public Property Get NewEnum() As IUnknown

Attribute NewEnum.VB_UserMemId = -4

'this property allows you to enumerate

'this collection with the For...Each syntax

Set NewEnum = mcolClones.[_NewEnum]

20 End Property

Public Sub Clear()

' empties the collection class

Set mcolClones = Nothing

25 Set mcolClones = New Collection

mblnIsDirty = True

End Sub

30 Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
ByVal lngEndIndex As Long)

Set mudtFile = New File

mudtFile.FileName = strFN

35 Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)

```

        Set mudtFile = Nothing
    End Sub

5   Public Sub ReadObjects()

        Dim udtClone As Clone

        On Error GoTo BeatIt

10      Do Until Err.Number <> 0

            Set udtClone = New Clone
            Call udtClone.ReadObjectData(mudtFile)
            udtClone.Index = NextID
15      Call mcolClones.Add(udtClone, Str(udtClone.Index))

        Loop

BeatIt:

20      Exit Sub

    End Sub

    Public Function WriteCollection(ByVal strFN As String, _
        ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

25      Set mudtFile = New File

        mudtFile.FileName = strFN
        WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

30      Set mudtFile = Nothing

        mblnIsDirty = False

    End Function

35  Public Sub WriteObjects()

        Dim udtClone As Clone

        For Each udtClone In mcolClones

```



```
    Call udtClone.WriteObjectData(mudtFile)
Next udtClone
```

```
End Sub
```

VBSCA -260-

```

' CConstraints.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "CConstraints"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    ' enable i/o
    Private mudtFile As New File

    'local variable to hold collection
15  Private mcolConstraint As Collection

    ' is dirty
    Private mblnIsDirty As Boolean

    Public Property Let IsDirty(ByVal blnNewValue As Boolean)

        mblnIsDirty = blnNewValue

20  End Property

    Public Property Get IsDirty() As Boolean

        Dim udtCon As Constraint

        For Each udtCon In mcolConstraint
25  If udtCon.IsDirty Then
            mblnIsDirty = True
            Exit For
        End If
        Next udtCon

30  IsDirty = mblnIsDirty

    End Property

    Private Sub Class_Initialize()

```

```
'creates the collection when this class is created
Set mcolConstraint = New Collection
```

```
End Sub
```

```
5 Private Sub Class_Terminate()
```

```
'destroys collection when this class is terminated
Set mcolConstraint = Nothing
```

```
End Sub
```

```
10 Public Property Get Item(vntIndexKey As Variant) As Constraint
```

```
'used when referencing an element in the collection
'vntIndexKey contains either the Index or Key to the collection,
'this is why it is declared as a Variant
'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
15 Set Item = mcolConstraint(vntIndexKey)
```

```
End Property
```

```
Public Property Get Count() As Long
```

```
'used when retrieving the number of elements in the
20 'collection. Syntax: Debug.Print x.Count
Count = mcolConstraint.Count
```

```
End Property
```

```
Public Sub AddObject(udtCon As Constraint)
```

```
25 ' adds constraint objects directly to the collection
```

```
udtCon.Index = NextID
Call mcolConstraint.Add(udtCon, Str(udtCon.Index))
```

```
mblnIsDirty = True
```

```
30 End Sub
```

```
Public Function Add(ByVal strConstraint As String, ByVal blnEnabled As Boolean, _
ByVal udtType As ConstraintType, ByVal strComment As String) As Constraint
```

```
35 'create a new object
```

```
Dim objNewMember As Constraint
Set objNewMember = New Constraint
```

```
'set the properties passed into the method
With objNewMember
```

```
5   .ConstraintString = strConstraint
    .Enabled = blnEnabled
    .ConstraintType = udtType
    .Comment = strComment
    .Index = NextID
```

```
10  ' add the new object to the collection
    Call mcolConstraint.Add(objNewMember, Str$(.Index))
```

```
End With
```

```
'return the object created
```

```
15  Set Add = objNewMember
```

```
Set objNewMember = Nothing
```

```
mblnIsDirty = True
```

```
End Function
```

```
20  Public Sub Remove(vntIndexKey As Variant)
```

```
'used when removing an element from the collection
```

```
'vntIndexKey contains either the Index or Key, which is why
```

```
'it is declared as a Variant
```

```
'Syntax: x.Remove(xyz)
```

```
25  mcolConstraint.Remove vntIndexKey
```

```
mblnIsDirty = True
```

```
End Sub
```

```
30  Public Function NewEnum() As IUnknown
```

```
Attribute NewEnum.VB_UserMemId = -4
```

```
Attribute NewEnum.VB_MemberFlags = "40"
```

```
'this property allows you to enumerate
```

```
'this collection with the For...Each syntax
```

```
35  Set NewEnum = mcolConstraint.[_NewEnum]
```

```
End Function
```

```
Private Function NextID() As Long
```

```
' creates a unique index to associate a constraint and the constraint listbox(es)
Static lngID As Long
```

```
lngID = lngID + 1
NextID = lngID
```

```
End Function
```

```
' returns true if strCon is already a constraint in the collection. Used
' when importing constraints to make sure dups are not introduced.
```

```
Public Function UniqueConstraint(ByVal strCon As String) As Boolean
```

```
Dim udtCon As Constraint
```

```
UniqueConstraint = True
```

```
' Check for duplicate constraint
For Each udtCon In mcolConstraint
    If strCon = udtCon.ConstraintString Then
        UniqueConstraint = False
        Exit For
    End If
Next udtCon
```

```
End Function
```

```
Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
    ByVal lngEndIndex As Long)
```

```
mudtFile.FileName = strFN
Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)
```

```
End Sub
```

```
Public Sub ReadObjects()
```

```
Dim udtCon As Constraint
```

```
On Error GoTo BeatIt
```

```
Do Until Err.Number <> 0
```

```
Set udtCon = New Constraint
Call udtCon.ReadObjectData(mudtFile)
udtCon.Index = NextID
```

Call mcolConstraint.Add(udtCon, Str(udtCon.Index))

Loop

5 BeatIt:

Exit Sub

End Sub

10 Public Function WriteCollection(ByVal strFN As String, _
ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

mudtFile.FileName = strFN

WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

mblnIsDirty = False

15 End Function

Public Sub WriteObjects()

Dim udtCon As Constraint

20 For Each udtCon In mcolConstraint

Call udtCon.WriteObjectData(mudtFile)

Next udtCon

25 End Sub

Public Sub Clear(ByVal udtType As VariableType)

' empties the collection class of all constraints of type udtType

Dim udtCon As Constraint

30 For Each udtCon In mcolConstraint

If udtCon.ConstraintType = udtType Then

Call mcolConstraint.Remove(Str(udtCon.Index))

End If

35 Next udtCon

End Sub

' returns true if an enabled string variable name was used
' in any enabled constraint

5 Public Function StringVarNamesUsed(ByVal udtCVar As CVariables) As Boolean

' First create a collection of all enabled constraint strings
Dim udtCon As Constraint
Dim colStrings As New Collection

10 For Each udtCon In mcolConstraint
 If udtCon.Enabled Then
 colStrings.Add udtCon.ConstraintString
 End If
Next udtCon

15 ' create a variable collection with variable names sorted in length
' from longest to shortest
Dim udtSCVar As CVariables

20 Set udtSCVar = udtCVar.SortVarNamesByLength

' nibble variable names out of the string collection, using enabled
' variable names sorted in length from longest to shortest
Dim vntS As Variant
25 Dim vntT As Variant
Dim vntStart As Variant
Dim udtVar As Variable

For Each vntS In colStrings
30 For Each udtVar In udtSCVar
 If udtVar.Enabled Then
 vntStart = InStr(1, vntS, udtVar.Name)
 If vntStart Then
 If udtVar.Typ = vtString Then
35 StringVarNamesUsed = True
Exit Function
 Else
 vntT = vntS
 vntS = left(vntT, vntStart - 1) & _
40 right(vntT, Len(vntT) - vntStart - _
Len(udtVar.Name) + 1)
 End If
 End If
End For

```
End If
Next udtVar
Next vntS
```

```
5 StringVarNamesUsed = False
```

```
End Function
```



```

' Checksum.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "Checksum"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private mcolStr As Collection

    Private Sub Class_Initialize()

        Set mcolStr = New Collection

15  End Sub

    Public Sub AddValue(ByVal strNewValue As String)

        Call mcolStr.Add(strNewValue)

    End Sub

    Public Function ComputeCS() As Double

20  Dim n As Integer
    Dim dblCS As Double
    Dim dblSum As Double
    Dim varStr As Variant
    Dim cntr As Integer
    Dim dblT As Double

25  cntr = 1

    ' On Error GoTo Overflow

30  For Each varStr In mcolStr
        dblSum = 0
        n = Len(varStr)
        While n > 0
35  dblSum = Asc(Mid(varStr, n, 1)) * n + dblSum

```

```

        n = n - 1
    Wend
    dblCS = dblSum * cntr + dblCS
    cntr = cntr + 1
5    Next varStr

'Overflow:

    ComputeCS = dblCS
10    Exit Function

End Function

```

```

' Clone.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "Clone"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    ' current version of data produced by this class
    Const mintVERSIONSTAMP As Integer = 1

    ' file name (without path) of this clone
15  Private mstrFN As String

    ' hold document handle
    Private mdocCloneDoc As Document

    ' checksum of variables
    Private mdblChecksum As Double

    ' index
20  Private mlngIndex As Long

    ' is dirty
    Private mblnIsDirty As Boolean

    ' has been routed to TCS
25  Private mbytIsRouted As Byte

    ' program
    Private mudtProgram As Program

    ' domain
    Private mudtDomain As Domain

    ' the batch id
30  Private mstrBatchID As String

    ' the target template
    Private udtDeliveryMode As DeliveryMode

```

' pure or real model
Private mudtNature As Nature

' TDer's estimate of difficulty (1-5)
Private mbytTDEstimate As Byte

5 ' difficulty has been calculated
Private mbytIsDifficultyCalculated As Byte

' the key
Private mstrKey As String

10 ' the item type
Private mudtItemType As ItemType

Public Enum Domain
doArithmetic = 0
doAlgebra = 1
doDataAnalysis = 2
doGeometry = 3
End Enum

Public Enum Nature
naPure = 0
naReal = 1
End Enum

' difficulty estimate
Private mudtDE As DifficultyEstimate

Private Sub Class_Initialize()

mblnIsDirty = False

End Sub

Public Property Get FileName() As String

FileName = mstrFN

End Property

Public Property Let FileName(ByVal strNewValue As String)

If mstrFN <> strNewValue Then

```
        mstrFN = strNewValue
        mblnIsDirty = True
    End If
```

```
End Property
```

```
5 Public Property Get CloneDoc() As Document
```

```
    Set CloneDoc = mdocCloneDoc
```

```
End Property
```

```
Public Property Let CloneDoc(ByVal docNewValue As Document)
```

```
10    Set mdocCloneDoc = docNewValue
```

```
End Property
```

```
Public Property Get Checksum() As Double
```

```
    Checksum = mdblChecksum
```

```
End Property
```

```
15 Public Property Let Checksum(ByVal dblNewValue As Double)
```

```
    If mdblChecksum <> dblNewValue Then
        mdblChecksum = dblNewValue
        mblnIsDirty = True
    End If
```

```
20 End Property
```

```
Public Property Get Index() As Long
```

```
    Index = mlngIndex
```

```
End Property
```

```
25 Public Property Let Index(ByVal lngNewValue As Long)
```

```
    If mlngIndex <> lngNewValue Then
        mlngIndex = lngNewValue
        mblnIsDirty = True
    End If
```

End Property

Public Property Get IsDirty() As Boolean

IsDirty = False

If IsDifficultyCalculated Then ' don't check DE if difficulty hasn't been calculated!

If mblnIsDirty Or mudtDE.IsDirty Then

IsDirty = True

End If

Else

If mblnIsDirty Then

IsDirty = True

End If

End If

End Property

Public Property Get IsRouted() As Byte

IsRouted = mbytIsRouted

End Property

Public Property Let IsRouted(ByVal bytNewValue As Byte)

If mbytIsRouted <> bytNewValue Then

mbytIsRouted = bytNewValue

mblnIsDirty = True

End If

End Property

Public Property Get Program() As Program

Program = mudtProgram

End Property

Public Property Let Program(ByVal udtNewValue As Program)

If mudtProgram <> udtNewValue Then

mudtProgram = udtNewValue

mblnIsDirty = True

End If

End Property

Public Property Get Domain() As Domain

Domain = muddDomain

5 End Property

Public Property Let Domain(ByVal udtNewValue As Domain)

If muddDomain <> udtNewValue Then

muddDomain = udtNewValue

mblnIsDirty = True

10 End If

End Property

Public Property Get IsDifficultyCalculated() As Byte

IsDifficultyCalculated = mbytIsDifficultyCalculated

15 End Property

Public Property Let IsDifficultyCalculated(ByVal bytNewValue As Byte)

If mbytIsDifficultyCalculated <> bytNewValue Then

mbytIsDifficultyCalculated = bytNewValue

mblnIsDirty = True

20 End If

End Property

Public Property Get TDEstimate() As Byte

TDEstimate = mbytTDEstimate

25 End Property

Public Property Let TDEstimate(ByVal bytNewValue As Byte)

If mbytTDEstimate <> bytNewValue Then

mbytTDEstimate = bytNewValue

mblnIsDirty = True

End If

End Property

Public Property Get BatchID() As String

BatchID = mstrBatchID

5 End Property

Public Property Let BatchID(ByVal strNewValue As String)

If mstrBatchID <> strNewValue Then

mstrBatchID = strNewValue

mblnIsDirty = True

10 End If

End Property

Public Property Get Key() As String

Key = mstrKey

End Property

15 Public Property Let Key(ByVal strNewValue As String)

If mstrKey <> strNewValue Then

mstrKey = strNewValue

mblnIsDirty = True

End If

20 End Property

Public Property Get ItemType() As ItemType

ItemType = mudtItemType

End Property

Public Property Let ItemType(ByVal udtNewValue As ItemType)

25 If mudtItemType <> udtNewValue Then

mudtItemType = udtNewValue

mblnIsDirty = True

End If

End Property

Public Property Get DeliveryMode() As DeliveryMode

DeliveryMode = udtDeliveryMode

5 End Property

Public Property Let DeliveryMode(ByVal udtNewValue As DeliveryMode)

If udtDeliveryMode <> udtNewValue Then

udtDeliveryMode = udtNewValue

mblnIsDirty = True

10 End If

End Property

Public Property Get Nature() As Nature

Nature = mudtNature

End Property

15 Public Property Let Nature(ByVal udtNewValue As Nature)

If mudtNature <> udtNewValue Then

mudtNature = udtNewValue

mblnIsDirty = True

End If

20 End Property

Public Property Get DiffEst() As DifficultyEstimate

Set DiffEst = mudtDE

End Property

Public Property Let DiffEst(ByVal udtNewValue As DifficultyEstimate)

25 Set mudtDE = udtNewValue

mblnIsDirty = True

End Property

Public Sub OpenDoc(ByVal udtWord As MSWord, ByVal strPath As String)

Dim udtDS As New DocStatus

If udtDS.IsOpen(mstrFN) = False Then

Set mdocCloneDoc = _

udtWord.WordApp.Documents.Open(Filename:=strPath & mstrFN)

End If

mdocCloneDoc.Activate

End Sub

Public Sub CloseDoc()

Dim udtDS As New DocStatus

If udtDS.IsOpen(mstrFN) Then

Call mdocCloneDoc.Close(wdSaveChanges) ' save changes

Set mdocCloneDoc = Nothing

End If

End Sub

Public Sub ReadObjectData(udtFile As File)

Dim vField As Variant

Call udtFile.ReadField(vField) ' returns the version stamp

Call udtFile.ReadField(vField)

FileName = ExtractFileName(vField)

Call udtFile.ReadField(vField)

Key = ExtractFileName(vField)

Call udtFile.ReadField(vField)

ItemType = ExtractFileName(vField)

Call udtFile.ReadField(vField)

Program = vField

Call udtFile.ReadField(vField)

Domain = vField

Call udtFile.ReadField(vField)

BatchID = vField

Call udtFile.ReadField(vField)

```

DeliveryMode = vField
Call udtFile.ReadField(vField)
Nature = vField
Call udtFile.ReadField(vField)
5 TDEstimate = vField
Call udtFile.ReadField(vField)
IsRouted = vField
Call udtFile.ReadField(vField)
IsDifficultyCalculated = vField
10 Set mudtDE = Nothing
If IsDifficultyCalculated Then
    Select Case Program
        Case prGRE
            Set mudtDE = New GREDifficultyEstimate
15 Case prGMAT
            Set mudtDE = New GMATDifficultyEstimate
        End Select
    Call mudtDE.ReadObjectData(udtFile)
End If

```

End Sub

Public Sub WriteObjectData(udtFile As File)

```

Call udtFile.WriteField(mintVERSIONSTAMP)
25 Call udtFile.WriteField(ExtractFileName(mstrFN))
Call udtFile.WriteField(Key)
Call udtFile.WriteField(ItemType)
Call udtFile.WriteField(Program)
Call udtFile.WriteField(Domain)
30 Call udtFile.WriteField(BatchID)
Call udtFile.WriteField(DeliveryMode)
Call udtFile.WriteField(Nature)
Call udtFile.WriteField(TDEstimate)
Call udtFile.WriteField(IsRouted)
35 Call udtFile.WriteField(IsDifficultyCalculated)
If IsDifficultyCalculated Then
    Call mudtDE.WriteObjectData(udtFile)
End If

```

40 mblnIsDirty = False

End Sub

```

' CModels.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "CModels"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    'to hold collection
    Private mcolModels As Collection

    Private Sub Class_Initialize()

15      'creates the collection when this class is created
        Set mcolModels = New Collection

    End Sub

    Private Sub Class_Terminate()

20      'destroys collection when this class is terminated
        Set mcolModels = Nothing

    End Sub

    Public Property Get Item(vntIndexKey As Variant) As Model

25      'used when referencing an element in the collection
        'vntIndexKey contains either the Index or Key to the collection,
        'this is why it is declared as a Variant
        'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
        Set Item = mcolModels(vntIndexKey)

30  End Property

    Public Property Get Count() As Long

        'used when retrieving the number of elements in the
        'collection. Syntax: Debug.Print x.Count
35  Count = mcolModels.Count

```

End Property

Public Sub AddObject(udtMod As Model)

5 ' adds model objects directly to the collection. Use the file name as the
 ' key.

 Call mcolModels.Add(udtMod, Str(udtMod.FileName))

End Sub

10 Public Function AddNew(ByVal strFN As String, _
 ByVal udtItemType As ItemType) As Model

 Dim udtMod As Model
 Dim udtSMC As SMCModel
 Dim udtQC As QCModel
15 Dim udtDS As DSModel

 Select Case udtItemType

 Case ptStandardMC
 Set udtSMC = New SMCModel
20 Set udtMod = udtSMC

 Case ptQuantComp
 Set udtQC = New QCModel
 Set udtMod = udtQC

25 Case ptDataSuff
 Set udtDS = New DSModel
 Set udtMod = udtDS

 End Select

30 ' file name has full path
 udtMod.FileName = strFN
 udtMod.IsFrozen = False

35 ' strip path from key
 Call mcolModels.Add(udtMod, ExtractFileName(strFN))

 Set AddNew = udtMod

End Function

Public Function AddExisting(ByVal strFN As String, _
ByVal udtItemType As ItemType) As Model

Dim udtMod As New Model
Dim udtSMC As SMCMModel
Dim udtQC As QCModel
Dim udtDS As DSModel

Select Case udtItemType

Case ptStandardMC
Set udtSMC = New SMCMModel
Set udtMod = udtSMC

Case ptQuantComp
Set udtQC = New QCModel
Set udtMod = udtQC

Case ptDataSuff
Set udtDS = New DSModel
Set udtMod = udtDS

End Select

' file name has full path
udtMod.FileName = strFN
Call udtMod.ReadModel

' strip path from key
Call mcolModels.Add(udtMod, ExtractFileName(strFN))

Set AddExisting = udtMod

End Function

Public Sub Remove(vntIndexKey As Variant)

'used when removing an element from the collection
'vntIndexKey contains either the Index or Key, which is why
'it is declared as a Variant
'Syntax: x.Remove(xyz)
mcolModels.Remove vntIndexKey

End Sub

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"
```

```
5      'this property allows you to enumerate
      'this collection with the For...Each syntax
      Set NewEnum = mcolModels.[_NewEnum]
```

```
End Property
```

```
Public Sub Clear()
```

```
10      ' empties the collection class
```

```
      Set mcolModels = Nothing
      Set mcolModels = New Collection
```

```
End Sub
```

```

' Constraint.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = 0 'False
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "Constraint"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
    Attribute VB_Ext_KEY = "Member0" ,"CloningConstraint"
    Attribute VB_Ext_KEY = "Member1" ,"DifficultyConstraint"
    Attribute VB_Ext_KEY = "Member2" ,"MathConstraint"
    Attribute VB_Ext_KEY = "Member3" ,"VariableDefinition"
20    Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
    Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

25    Private mudtType As VariableType
    Private mstrConstraint As String
    Private mstrComment As String
    Private mlngIndex As Long
    Private mblnEnabled As Boolean
    Private mblnIsDirty As Boolean

30    ' These numbers correspond to the indices of the constraint listboxes in frmTCA
    Public Enum ConstraintType
        ctVariation = 0
        ctDistractor = 1
    End Enum

35    Public Property Get ConstraintString() As String

        ConstraintString = mstrConstraint

    End Property

```



```
Public Property Let ConstraintString(ByVal strNewValue As String)
```

```
    If mstrConstraint <> strNewValue Then  
        mstrConstraint = strNewValue  
        mblnIsDirty = True  
5    End If
```

```
End Property
```

```
Public Property Get Comment() As String
```

```
    Comment = mstrComment
```

```
10 End Property
```

```
Public Property Let Comment(ByVal strNewValue As String)
```

```
    If mstrComment <> strNewValue Then  
        mstrComment = strNewValue  
        mblnIsDirty = True  
15    End If
```

```
End Property
```

```
Public Property Get ConstraintType() As ConstraintType
```

```
20    ConstraintType = mudtType
```

```
End Property
```

```
Public Property Let ConstraintType(ByVal udtNewValue As ConstraintType)
```

```
    If mudtType <> udtNewValue Then  
        mudtType = udtNewValue  
        mblnIsDirty = True  
25    End If
```

```
End Property
```

```
30 Public Property Get index() As Long
```

```
    index = mlngIndex
```

```
End Property
```

Public Property Let index(ByVal lngNewValue As Long)

 mLngIndex = lngNewValue

End Property

5 Public Property Get Enabled() As Boolean

 Enabled = mblnEnabled

End Property

Public Property Let Enabled(ByVal blnNewValue As Boolean)

10 If mblnEnabled <> blnNewValue Then
 mblnEnabled = blnNewValue
 mblnIsDirty = True
End If

15 End Property

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

 mblnIsDirty = blnNewValue

End Property

20 Public Property Get IsDirty() As Boolean

 IsDirty = mblnIsDirty

End Property

25 Public Sub Update(ByVal strConstraint As String, ByVal udtType As ConstraintType, _
 ByVal strComment As String)

 ConstraintString = strConstraint

 ConstraintType = udtType

 Comment = strComment

30 End Sub

Public Sub ReadObjectData(udtFile As File)

 Dim vField As Variant

```
Call udtFile.ReadField(vField) ' read version stamp
Call udtFile.ReadField(vField)
ConstraintType = vField
```

```
5 Call udtFile.ReadField(vField)
Enabled = vField
```

```
Call udtFile.ReadField(vField)
ConstraintString = vField
```

```
10 Call udtFile.ReadField(vField)
Comment = vField
```

```
End Sub
```

```
15 Public Sub WriteObjectData(udtFile As File)
```

```
Call udtFile.WriteField(mintVERSIONSTAMP)
Call udtFile.WriteField(ConstraintType)
Call udtFile.WriteField(Enabled)
Call udtFile.WriteField(ConstraintString)
20 Call udtFile.WriteField(Comment)
```

```
mblnIsDirty = False
```

```
End Sub
```

```
' makes a copy of this object
```

```
25 Public Function Copy() As Constraint
```

```
Dim udtC As New Constraint
```

```
udtC.Enabled = Enabled
```

```
udtC.index = index
```

```
30 udtC.IsDirty = IsDirty
```

```
udtC.ConstraintType = ConstraintType
```

```
udtC.ConstraintString = ConstraintString
```

```
udtC.Comment = Comment
```

```
35 Set Copy = udtC
```

```
End Function
```



```

' ConstraintSolver.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "ConstraintSolver"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    Private mcolVs As Collection
    Private mcolVsSave As Collection
    Private mcolCs As Collection
    Private mcolCsSave As Collection
20    Private mcolValues As Collection
    Private mbytDiffWeight As Byte
    Private mdblChecksum As Double
    Private mintIndex As Integer

    Private WithEvents mwudtP As Prolog
25    Attribute mwudtP.VB_VarHelpID = -1
    Private mlngRet As Long

    Private mblnPrologIsRunning As Boolean

    Public Enum SolveRequester
        srTest = 0
30        srGenerate = 1
    End Enum

    Public Enum SolveReturn
        srNoSolutions = 0
        srSuccess = 1
35        srPrologAborted = -1
        srPrologError = -2
    End Enum

    Private mudtSolveRequester As SolveRequester

```

```

Private Sub Class_Initialize()

    Set mcolVs = New Collection
    Set mcolVsSave = New Collection
    Set mcolCs = New Collection
5    Set mcolCsSave = New Collection
    Set mcolValues = New Collection

End Sub

Private Sub Class_Terminate()

10    ' Kill Prolog
    Set mwudtP = Nothing

End Sub

Public Property Let Prolog(ByVal udtNewValue As Prolog)

    Set mwudtP = udtNewValue
15 End Property

Public Property Let DiffWeight(ByVal bytNewValue As Byte)

    mbytDiffWeight = bytNewValue

End Property

20 Public Sub AddVariable(ByVal udtNewValue As Variable)

    If udtNewValue.Enabled Then
        Call mcolVs.Add(udtNewValue.Copy) ' uses a copy of the variable
        Call mcolVsSave.Add(udtNewValue.Copy)
    End If

25 End Sub

Public Sub AddConstraint(ByVal udtNewValue As Constraint)

    If udtNewValue.Enabled Then
        Call mcolCs.Add(udtNewValue.Copy) ' uses a copy of the constraint
30    Call mcolCsSave.Add(udtNewValue.Copy) '
    End If

```

End Sub

Public Function GetNextValue(strVarName As String, _
strValue As String) As Boolean

Dim udtVal As Value

If mintIndex <= mcolValues.Count Then

Set udtVal = mcolValues.Item(mintIndex)

strVarName = udtVal.VariableName

strValue = udtVal.Value

' if the value is ^, replace with ^^ so Word doesn't choke

If strValue = "^" Then strValue = "^^"

mintIndex = mintIndex + 1

GetNextValue = True

Else

GetNextValue = False

End If

End Function

Public Sub ResetValueIndex()

mintIndex = 1

End Sub

Public Property Get Checksum()

Checksum = mdb1Checksum

End Property

Public Function Solve(ByVal udtSolveRequester As SolveRequester) As SolveReturn

Dim udtVal As Value

Dim udtC As Constraint

Dim udtV As Variable

Dim udtVS As VarString

Dim udtSS As StringSolver

mudtSolveRequester = udtSolveRequester

Set mcolValues = New Collection

mintIndex = 1

CreateValueCollection

If mcolValues.Count = 0 Then

Solve = srNoSolutions

Exit Function

End If

' solve all string variables

For Each udtV In mcolVs

If udtV.Type = vtString Then

Set udtVS = udtV

' if this variable has no strings, error

If udtVS.StringCollection.Count = 0 Then

Solve = srNoSolutions

Exit Function

End If

Set udtSS = New StringSolver

udtSS.StringVariable = udtVS

Call LoadStringValue(udtVS, udtSS)

End If

Next udtV

' resolve any nested values for all string variable names

ResolveNestedStrings

' resolve string variable names embedded in math variable ranges

ResolveStringsInMathVariables

' resolve string variable names embedded in constraints

ResolveConstraints

' set the difference weight (difference between variants)

mwudtP.DiffWeight = mbytdiffWeight

Dim blnMathToSolve As Boolean

' add non-string variables to prolog via the value object collection

For Each udtVal In mcolValues

If Not udtVal.VariableType = vtString Then

Call mwudtP.AddVariable(udtVal.PrologString)

blnMathToSolve = True

End If

Next udtVal


```

' add all constraints
For Each udtC In mcolCs
    Call mwudtP.AddConstraint(udtC.ConstraintString)
    blnMathToSolve = True
5 Next udtC

' call prolog if there are math constraints, error if no solution found
If blnMathToSolve Then
    ' get rid of the kill file if it exists
10 DestroyKillFile
    mblnPrologIsRunning = True
    ' runs async, notifies this class when it's done via the Finished event
    mwudtP.SolveConstraintsRandomly
    If udtSolveRequester = srTest Then
15 frmProlog.Caption = "Testing constraints"
        frmProlog.lblProlog.Caption = "Click Abort to terminate this test."
        frmProlog.Show vbModal
    Else
        Do
20         DoEvents
        Loop While mblnPrologIsRunning
    End If
    If frmProlog.Abort Then
        ' create the kill file
25 CreateKillFile
        Solve = srPrologAborted
        Exit Function
    End If
    ' not aborted
30 Select Case mlngRet
        Case Is < 0
            Solve = srPrologError
            Call MsgBox("Prolog error: " & Str(mlngRet), vbExclamation, "Error")
            Exit Function
        Case 0
35         Solve = srNoSolutions
            Exit Function
    End Select
End If
40

' load up values from Prolog
For Each udtVal In mcolValues
    If Not udtVal.VariableType = vtString Then
        udtVal.Value = mwudtP.Value(udtVal.VariableName)
45 End If

```

Next udtVal

' resolve string values that are math variable names
ResolveMathVariablesInStrings

Dim udtChecksum As New Checksum

5 ' compute the checksum of values
For Each udtVal In mcolValues
 If udtVal.Checksum Then
 Call udtChecksum.AddValue(udtVal.Value)

10 End If
Next udtVal

mdblChecksum = udtChecksum.ComputeCS

Solve = srSuccess

15 ' restore the variable and constraint collections their original states,
 ' as substitutions may have contaminated them.
Set mcolVs = New Collection
Set mcolCs = New Collection

20 For Each udtV In mcolVsSave
 Call mcolVs.Add(udtV.Copy)
Next udtV

25 For Each udtC In mcolCsSave
 Call mcolCs.Add(udtC.Copy)
Next udtC

End Function

' this event raised in Prolog class
Private Sub mwudtP_Finished(ByVal lngRet As Long)

30 mblnPrologIsRunning = False
 mlngRet = lngRet

 ' kill the form if this is a test
 If mudtSolveRequester = srTest Then
35 frmProlog.Kill
 End If

End Sub

Private Sub CreateValueCollection()

Dim intI As Integer
Dim udtV As Variable
Dim udtVS As VarString
5 Dim udtVal As Value

For Each udtV In mcolVs

If udtV.Type = vtString Then

Set udtVS = udtV

10 If udtVS.IsIndexed Then

For intI = udtVS.NumIndices To 1 Step -1

Set udtVal = New Value

udtVal.VariableName = GetIndexedName(udtV.name, intI)

udtVal.VariableType = udtV.Type

15 udtVal.Checksum = udtV.Checksum

udtVal.PrologString = udtV.PrologFormat

Call mcolValues.Add(udtVal, udtVal.VariableName)

Next intI

Else

20 Set udtVal = New Value

udtVal.VariableName = udtV.name

udtVal.VariableType = udtV.Type

udtVal.Checksum = udtV.Checksum

udtVal.PrologString = udtV.PrologFormat

25 Call mcolValues.Add(udtVal, udtVal.VariableName)

End If

Else

Set udtVal = New Value

udtVal.VariableName = udtV.name

30 udtVal.VariableType = udtV.Type

udtVal.Checksum = udtV.Checksum

udtVal.PrologString = udtV.PrologFormat

Call mcolValues.Add(udtVal, udtVal.VariableName)

End If

35 Next udtV

End Sub

Private Sub LoadStringValue(ByVal udtV As Variable, _
ByVal udtSS As StringSolver)

40 Dim intI As Integer
Dim varS As Variant
Dim strVN As String

```
Dim udtVal As Value
Dim udtVS As VarString
```

```
Set udtVS = udtV
```

```
' get the value or values (if indexed)
```

```
If udtVS.IsIndexed Then
```

```
    intI = 1
```

```
    For Each varS In udtSS.RandomValueCollection
```

```
        strVN = GetIndexedName(udtV.name, intI)
```

```
        Set udtVal = mcolValues.Item(strVN)
```

```
        udtVal.Value = varS
```

```
        intI = intI + 1
```

```
    Next varS
```

```
Else
```

```
    Set udtVal = mcolValues.Item(udtV.name)
```

```
    udtVal.Value = udtSS.RandomValueCollection(1)
```

```
End If
```

```
End Sub
```

```
Private Sub ResolveNestedStrings()
```

```
    Dim blnContinue As Boolean
```

```
    Dim udtVal As Value
```

```
    Do
```

```
        blnContinue = False
```

```
        For Each udtVal In mcolValues
```

```
            If udtVal.VariableType = vtString Then
```

```
                If ResolveString(udtVal.VariableName) Then
```

```
                    blnContinue = True
```

```
                End If
```

```
            End If
```

```
        Next udtVal
```

```
    Loop Until blnContinue = False
```

```
End Sub
```

```
Private Function ResolveString(ByVal strVN As String) As Boolean
```

```
    Dim udtVal As Value
```

```
    Dim udtVal2 As Value
```

```
    Dim strT As String
```

ResolveString = False

For Each udtVal In mcolValues

If udtVal.VariableType = vtString Then

Set udtVal2 = mcolValues.Item(strVN)

strT = ReplaceAll(udtVal.Value, strVN, udtVal2.Value)

If strT <> udtVal.Value Then

udtVal.Value = strT

ResolveString = True

End If

End If

Next udtVal

End Function

Private Sub ResolveStringsInMathVariables()

Dim udtVal As Value

Dim udtVal2 As Value

For Each udtVal In mcolValues

If udtVal.VariableType = vtString Then

For Each udtVal2 In mcolValues

If Not udtVal2.VariableType = vtString Then

udtVal2.PrologString = ReplaceAll(udtVal2.PrologString, _
udtVal.VariableName, udtVal.Value)

End If

Next udtVal2

End If

Next udtVal

End Sub

Private Sub ResolveConstraints()

Dim udtC As Constraint

Dim udtVal As Value

For Each udtVal In mcolValues

If udtVal.VariableType = vtString Then

For Each udtC In mcolCs

udtC.ConstraintString = ReplaceAll(udtC.ConstraintString, _
udtVal.VariableName, udtVal.Value)

Next udtC

End If

Next udtVal

End Sub

Private Sub ResolveMathVariablesInStrings()

Dim udtVal As Value

Dim udtVal2 As Value

For Each udtVal In mcolValues

If udtVal.VariableType = vtString Then

For Each udtVal2 In mcolValues

If Not udtVal2.VariableType = vtString Then

udtVal.Value = ReplaceAll(udtVal.Value, udtVal2.VariableName, _
udtVal2.Value)

End If

Next udtVal2

End If

Next udtVal

End Sub

' CVariables.cls

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB_Name = "CVariables"

Attribute VB_GlobalNameSpace = False

Attribute VB_Creatable = True

Attribute VB_PredeclaredId = False

Attribute VB_Exposed = False

Attribute VB_Ext_KEY = "SavedWithClassBuilder", "Yes"

Attribute VB_Ext_KEY = "Collection", "Variable"

Attribute VB_Ext_KEY = "Member0", "Variable"

Attribute VB_Ext_KEY = "Top_Level", "Yes"

Option Explicit

' enable i/o

Private mudtFile As File

'to hold collection

Private mcolVariable As Collection

```

' is dirty
Private mblnIsDirty As Boolean

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

    mblnIsDirty = blnNewValue

```

```

5 End Property

```

```

Public Property Get IsDirty() As Boolean

```

```

    Dim udtVar As Variable

```

```

    For Each udtVar In mcolVariable

```

```

10     If udtVar.IsDirty Then

```

```

        mblnIsDirty = True

```

```

        Exit For

```

```

    End If

```

```

Next udtVar

```

```

15 IsDirty = mblnIsDirty

```

```

End Property

```

```

Private Sub Class_Initialize()

```

```

    'creates the collection when this class is created

```

```

20 Set mcolVariable = New Collection

```

```

    Set mudtFile = New File

```

```

End Sub

```

```

25 Private Sub Class_Terminate()

```

```

    'destroys collection when this class is terminated

```

```

    Set mcolVariable = Nothing

```

```

    'destroys the File object

```

```

30 Set mudtFile = Nothing

```

```

End Sub

```

```

Public Property Get Item(vntIndexKey As Variant) As Variable

```

```

'used when referencing an element in the collection
'vntIndexKey contains either the Index or Key to the collection,
'this is why it is declared as a Variant
'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
5 Set Item = mcolVariable(vntIndexKey)

```

```
End Property
```

```
Public Property Get Count() As Long
```

```

'used when retrieving the number of elements in the
10 'collection. Syntax: Debug.Print x.Count
Count = mcolVariable.Count

```

```
End Property
```

```
15 Public Sub AddObject(udtVar As Variable)
```

```
' adds variable objects directly to the collection
```

```

udtVar.Index = NextID
Call mcolVariable.Add(udtVar, Str(udtVar.Index))

```

```
20 End Sub
```

```

Public Function AddInteger(ByVal strName As String, ByVal blnEnabled As Boolean, _
ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean) As Variable

```

```

'create a new object
25 Dim udtVar As Variable
Dim udtVarInteger As New VarInteger

```

```
Set udtVar = udtVarInteger
```

```

'set the properties passed into the method
30 With udtVar

```

```

.Type = vtInteger
.Name = strName
.Enabled = blnEnabled
.Index = NextID
35 .Checksum = blnChecksum

```

```
End With
```

```
With udtVarInteger
```



```

        .From = strFrom
        .Too = strTo
        .By = strBy
        .IsIndependent = blnIsIndependent
5      End With

      ' add the new object to the collection
      Call mcolVariable.Add(udtVarInteger, Str(udtVar.Index))

      'return the object created
10     Set AddInteger = udtVarInteger

End Function

Public Function AddReal(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
15    ByVal blnTrailingZeros As Boolean, _
    ByVal strPrecision As String, ByVal blnOnGrid As Boolean) As Variable

    'create a new object
    Dim udtVar As Variable
20    Dim udtVarReal As New VarReal

    Set udtVar = udtVarReal

    'set the properties passed into the method
    With udtVar
25        .Typ = vtReal
        .Name = strName
        .Enabled = blnEnabled
        .Index = NextID
        .Checksum = blnChecksum
30    End With

    With udtVarReal
        .From = strFrom
        .Too = strTo
35        .By = strBy
        .IsIndependent = blnIsIndependent
        .TrailingZeros = blnTrailingZeros
        .Precision = strPrecision
        .IsOnGrid = blnOnGrid
40    End With

```

```
' add the new object to the collection
Call mcolVariable.Add(udtVarReal, Str(udtVar.Index))
```

```
'return the object created
Set AddReal = udtVarReal
```

5 End Function

```
Public Function AddFraction(ByVal strName As String, ByVal blnEnabled As Boolean, _
    ByVal strFromNum As String, ByVal strFromDen As String, _
    ByVal strToNum As String, ByVal strToDen As String, _
    ByVal strByNum As String, ByVal strByDen As String, _
10 ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
    ByVal blnMixedNumbers As Boolean) As Variable
```

```
'create a new object
Dim udtVar As Variable
15 Dim udtVarFraction As New VarFraction
```

```
Set udtVar = udtVarFraction
```

```
'set the properties passed into the method
With udtVar
```

```
20 .Typ = vtFraction
    .Name = strName
    .Enabled = blnEnabled
    .Index = NextID
    .Checksum = blnChecksum
```

```
25 End With
```

```
With udtVarFraction
    .FromNumerator = strFromNum
    .FromDenominator = strFromDen
30 .ToNumerator = strToNum
    .ToDenominator = strToDen
    .ByNumerator = strByNum
    .ByDenominator = strByDen
    .IsIndependent = blnIsIndependent
35 .MixedNumbers = blnMixedNumbers
End With
```

```
' add the new object to the collection
Call mcolVariable.Add(udtVarFraction, Str(udtVar.Index))
```

40 'return the object created

Set AddFraction = udtVarFraction

End Function

Public Function AddString(ByVal strName As String, ByVal blnEnabled As Boolean, _
ByVal blnChecksum As Boolean, ByVal strDelimiter As String, _
5 ByVal blnIsIndexed As Boolean, ByVal colString As Collection) As Variable

'create a new object

Dim udtVar As Variable

Dim udtVarString As New VarString

10 Set udtVar = udtVarString

'set the properties passed into the method

With udtVar

 .Typ = vtString

 .Name = strName

 .Enabled = blnEnabled

 .Index = NextID

 .Checksum = blnChecksum

End With

20 udtVarString.Delimiter = strDelimiter

 udtVarString.StringCollection = colString

 udtVarString.IsIndexed = blnIsIndexed

25 ' add the new object to the collection

 Call mcolVariable.Add(udtVarString, Str(udtVar.Index))

'return the object created

Set AddString = udtVarString

End Function

30 Public Function AddUntyped(ByVal strName As String, ByVal blnEnabled As Boolean, _
 ByVal blnChecksum As Boolean)

'create a new object

Dim udtVar As Variable

35 Dim udtVarUntyped As New VarUntyped

Set udtVar = udtVarUntyped

'set the properties passed into the method

```

With udtVar
    .Typ = vtUntyped
    .Name = strName
    .Enabled = blnEnabled
5    .Index = NextID
    .Checksum = blnChecksum
End With

' add the new object to the collection
10 Call mcolVariable.Add(udtVarUntyped, Str(udtVar.Index))

'return the object created
Set AddUntyped = udtVarUntyped

End Function

Public Sub Remove(vntIndexKey As Variant)

15 'used when removing an element from the collection
'vntIndexKey contains either the Index or Key, which is why
'it is declared as a Variant
'Syntax: x.Remove(xyz)
mcolVariable.Remove vntIndexKey

20 mblnIsDirty = True

End Sub

Public Property Get NewEnum() As IUnknown
25 Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"

'this property allows you to enumerate
'this collection with the For...Each syntax
Set NewEnum = mcolVariable.[_NewEnum]

30 End Property

Private Function NextID() As Long

't creates a unique index to associate a variable and the variable listbox
Static lngID As Long

35 lngID = lngID + 1

```

NextID = lngID

End Function

' returns true if strName is already a variable name in the collection. If the
5 ' optional parameter is used, the function will not check that variable for a dup.

Public Function UniqueName(ByVal strName As String, _
Optional ByVal bytSkipThisVar As Byte = 0, _
Optional ByVal udtSkipVar As Variable) As Boolean

Dim udtVar As Variable

UniqueName = True

' Check for duplicate variable name
For Each udtVar In mcolVariable

15 If UCASE(strName) = UCASE(udtVar.Name) Then
If bytSkipThisVar = 1 Then
If udtSkipVar.Index <> udtVar.Index Then
UniqueName = False
20 Exit For
End If
Else
UniqueName = False
Exit For
25 End If
End If

Next udtVar

30 End Function

' Check enabled variables in collection for duplicate names.

Public Function DuplicateNames() As Boolean

Dim udtVar1 As Variable
Dim udtVar2 As Variable
35 Dim intI1 As Integer
Dim intI2 As Integer

DuplicateNames = False

```

For intI1 = 1 To mcolVariable.Count
  For intI2 = 1 To mcolVariable.Count
    If intI1 <> intI2 Then
      Set udtVar1 = mcolVariable.Item(intI1)
      Set udtVar2 = mcolVariable.Item(intI2)
      If udtVar1.Enabled And udtVar2.Enabled Then
        If udtVar1.Name = udtVar2.Name Then
          DuplicateNames = True
          Exit Function
        End If
      End If
    End If
  Next intI2
Next intI1

End Function

Public Sub ReadCollection(ByVal strFN As String, ByVal lngStartIndex As Long, _
  ByVal lngEndIndex As Long)

  mudtFile.FileName = strFN
  Call mudtFile.ReadFile(Me, lngStartIndex, lngEndIndex)

End Sub

Public Sub ReadObjects()

  Dim udtVar As Variable
  Dim udtType As VariableType

  On Error GoTo BeatIt

  Do Until Err.Number <> 0

    Set udtVar = New Variable
    udtType = udtVar.ReadType(mudtFile)

    Select Case udtType
      Case vtInteger
        Set udtVar = New VarInteger
        udtVar.Typ = vtInteger
      Case vtReal
        Set udtVar = New VarReal
        udtVar.Typ = vtReal
      Case vtFraction

```

```

        Set udtVar = New VarFraction
        udtVar.Typ = vtFraction
    Case vtString
        Set udtVar = New VarString
        udtVar.Typ = vtString
    Case vtUntyped
        Set udtVar = New VarUntyped
        udtVar.Typ = vtUntyped
End Select

    Call udtVar.ReadObjectData(mudtFile)
    udtVar.Index = NextID
    Call mcolVariable.Add(udtVar, Str(udtVar.Index))

```

```

Loop

```

```

BeatIt:
    Exit Sub

```

```

End Sub

```

```

Public Function WriteCollection(ByVal strFN As String, _
    ByVal lngIndexPos As Long, ByVal lngSeekPos) As Long

    mudtFile.FileName = strFN
    WriteCollection = mudtFile.WriteFile(Me, False, lngIndexPos, lngSeekPos)

```

```

    mblnIsDirty = False

```

```

End Function

```

```

Public Sub WriteObjects()

```

```

    Dim udtVar As Variable

```

```

    For Each udtVar In mcolVariable
        Call udtVar.WriteObjectData(mudtFile)
    Next udtVar

```

```

End Sub

```

```

Public Sub Clear()

```

```

    ' empties the collection class

```

```
Set mcolVariable = Nothing
Set mcolVariable = New Collection
```

```
End Sub
```

```
5 ' returns a collection of variables sorted by length of variable name,
' longest to shortest
```

```
Public Function SortVarNamesByLength() As CVariables
```

```
Dim udtVar As Variable
Dim intLen As Integer
10 Dim intLongest As Integer
Dim udtCVar As New CVariables
```

```
' Find longest variable name
For Each udtVar In mcolVariable
```

```
15 If udtVar.Enabled Then
    intLen = Len(udtVar.Name)
    If intLen > intLongest Then
        intLongest = intLen
    End If
```

```
20 End If
Next udtVar
```

```
' Sort variables by length of name - longest first
Do
```

```
25 For Each udtVar In mcolVariable
    If udtVar.Enabled Then
        intLen = Len(udtVar.Name)
        If intLen = intLongest Then
            ' Put this var in sorted collection
            udtCVar.AddObject udtVar
        End If
```

```
30 End If
Next udtVar
intLongest = intLongest - 1
Loop While intLongest > 0
```

```
35 Set SortVarNamesByLength = udtCVar
```

```
End Function
```



```
' CVariants.cls
VERSION 1.0 CLASS
BEGIN
```

```
MultiUse = -1 'True
```

```
END
```

```
Attribute VB_Name = "CVariants"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = True
```

```
Attribute VB_PredeclaredId = False
```

```
Attribute VB_Exposed = False
```

```
Option Explicit
```

```
'to hold collection
```

```
Private mcolVariants As Collection
```

```
Private Sub Class_Initialize()
```

```
'creates the collection when this class is created
```

```
Set mcolVariant = New Collection
```

```
End Sub
```

```
Private Sub Class_Terminate()
```

```
'destroys collection when this class is terminated
```

```
Set mcolVariant = Nothing
```

```
End Sub
```

```
Public Property Get Item(vntIndexKey As Variant) As Variant
```

```
'used when referencing an element in the collection
```

```
'vntIndexKey contains either the Index or Key to the collection,
```

```
'this is why it is declared as a Variant
```

```
'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
```

```
Set Item = mcolVariant(vntIndexKey)
```

```
End Property
```

```
Public Property Get Count() As Long
```

'used when retrieving the number of elements in the
'collection. Syntax: Debug.Print x.Count
Count = mcolVariant.Count

5 End Property

Public Sub AddObject(udtVar As Variant)

' adds variable objects directly to the collection

udtVar.Index = NextID

10 Call mcolVariant.Add(udtVar, Str(udtVar.Index))

End Sub

Public Function Add(ByVal strName As String, _
ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String) As Variant

15 'create a new object
Dim udtVar As Variant
Dim udtVarInteger As New VarInteger

Set udtVar = udtVarInteger

20 'set the properties passed into the method
With udtVar
.Name = strName
.Index = NextID
End With

25 With udtVarInteger
.From = strFrom
.Too = strTo
.By = strBy

30 End With

' add the new object to the collection
Call mcolVariant.Add(udtVarInteger, Str(udtVar.Index))

35 'return the object created
Set AddInteger = udtVarInteger

End Function

```
Public Sub Remove(vntIndexKey As Variant)
```

```
    'used when removing an element from the collection
```

```
    'vntIndexKey contains either the Index or Key, which is why
```

```
    'it is declared as a Variant
```

```
5    'Syntax: x.Remove(xyz)
```

```
    mcolVariant.Remove vntIndexKey
```

```
End Sub
```

```
Public Property Get NewEnum() As IUnknown
```

```
10    'this property allows you to enumerate
```

```
    'this collection with the For...Each syntax
```

```
    Set NewEnum = mcolVariant.[_NewEnum]
```

```
End Property
```

```
15 Private Function NextID() As Long
```

```
    ' creates a unique index to associate a variable and the variable listbox
```

```
    Static lngID As Long
```

```
    lngID = lngID + 1
```

```
20    NextID = lngID
```

```
End Function
```

```
Public Sub Clear()
```

```
    ' empties the collection class
```

```
25    Set mcolVariant = Nothing
```

```
    Set mcolVariant = New Collection
```

```
End Sub
```

```

' DifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "DifficultyEstimate"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private mblnIsDirty As Boolean

    Private Sub Class_Initialize()

        mblnIsDirty = False
15  End Sub

    Public Property Let IsDirty(ByVal blnNewValue As Boolean)

        mblnIsDirty = blnNewValue
20  End Property

    Public Property Get IsDirty() As Boolean

        IsDirty = mblnIsDirty

    End Property

    ' implemented in the subclasses of DifficultyEstimate
25  Public Function ComputeDifficulty() As Double

    End Function

    ' implemented in the subclasses of DifficultyEstimate
    Public Function Copy() As DifficultyEstimate

    End Function

30  ' implemented in the subclasses of DifficultyEstimate
    Public Sub ReadObjectData(udtFile As File)

```

End Sub

' implemented in the subclasses of DifficultyEstimate
Public Sub WriteObjectData(udtFile As File)

End Sub

VBSCA -312-

```

' DocStatus.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "DocStatus"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

' returns true if this document strFN is open
Public Function IsOpen(ByVal strFN As String) As Boolean

    Dim docD As Document
15
    For Each docD In Documents
        If InStr(1, strFN, docD.Name) Then
            IsOpen = True
            Exit Function
20        End If
    Next docD

    IsOpen = False

End Function

```

```

' DSMODEL.CLS
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "DSModel"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    Implements Model

    Dim mudtModel As Model

    Private Sub Class_Initialize()

        Set mudtModel = New Model
20    End Sub

    ' Delegated to Class Model
    Public Property Get Model_FileName() As String

        Model_FileName = mudtModel.FileName

    End Property

25    ' Delegated to Class Model
    Public Property Let Model_FileName(ByVal strNewValue As String)

        mudtModel.FileName = strNewValue

    End Property

    ' Delegated to Class Model
30    Public Property Get Model_IsFrozen() As Boolean

        Model_IsFrozen = mudtModel.IsFrozen

```

End Property

' Delegated to Class Model

Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)

 mudtModel.IsFrozen = blnNewValue

5 End Property

' Delegated to Class Model

Public Property Get Model_Comments() As String

 Model_Comments = mudtModel.Comments

End Property

10 ' Delegated to Class Model

Public Property Let Model_Comments(ByVal strNewValue As String)

 mudtModel.Comments = strNewValue

End Property

' Delegated to Class Model

15 Public Property Get Model_Clones() As CClones

 Set Model_Clones = mudtModel.Clones

End Property

' Delegated to Class Model

Public Property Get Model_Variables() As CVariables

20 Set Model_Variables = mudtModel.Variables

End Property

' Delegated to Class Model

Public Property Get Model_Constraints() As CConstraints

 Set Model_Constraints = mudtModel.Constraints

25 End Property

'Delegated to Class Model


```
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)
```

```
    Call mudtModel.AddChecksum(dblChecksum)
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_InitChecksums()
```

```
    mudtModel.InitChecksums
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_InitTempChecksums()
```

```
    mudtModel.InitTempChecksums
```

```
End Sub
```

```
'Delegated to Class Model
```

```
Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean
```

```
    Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)
```

```
End Function
```

```
' Delegated to Class Model
```

```
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)
```

```
    mudtModel.IsDirty = blnNewValue
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Property Get Model_IsDirty() As Boolean
```

```
    Model_IsDirty = mudtModel.IsDirty
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Property Let Model_LastClone(ByVal intNewValue As Integer)
```

```
    mudtModel.LastClone = intNewValue
```

End Property

' Delegated to Class Model

Public Property Get Model_LastClone() As Integer

Model_LastClone = mudtModel.LastClone

5 End Property

' Delegated to Class Model

Public Sub Model_FreezeModel()

Call mudtModel.FreezeModel

End Sub

10

' Delegated to Class Model

Public Sub Model_OpenDoc(ByVal udtWord As MSWord)

Call mudtModel.OpenDoc(udtWord)

End Sub

' Delegated to Class Model

15

Public Sub Model_CloseDoc()

Call mudtModel.CloseDoc

End Sub

' Delegated to Class Model

Public Sub Model_CloseAllCloneDocs()

20

Call mudtModel.CloseAllCloneDocs

End Sub

' Delegated to Class Model

Public Sub Model_ReadModel()

mudtModel.ReadModel

25

End Sub

' Delegated to Class Model

```
Public Sub Model_ReadObjects()
```

```
    mudtModel.ReadObjects
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_WriteModel()
```

```
    mudtModel.WriteModel
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_WriteObjects()
```

```
    mudtModel.WriteObjects
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
```

```
    ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
```

```
    blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean
```

```
    Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _  
        blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)
```

```
End Function
```

```
' implemented here
```

```
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _  
    ByVal intNumClones As Integer, ByVal bytDifference As Byte)
```

```
    Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _  
        bytDifference, 285)
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_SubstituteValues(ByVal objO As Object, _
```

```
    ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
```

```
    ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
```

```
    ByVal intStartPos As Integer)
```

End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

Dim rnumber As Integer
Dim statementRange As Range
5 Dim firstNSE As String
Dim secondNSE As String

With udtClone.CloneDoc

rnumber = .Tables(1).Rows.Count * Rnd + 0.5
.Tables(1).Cell(Row:=rnumber, Column:=1).Range.Copy
10 firstNSE = .Tables(1).Cell(Row:=rnumber, Column:=2).Range.Text
firstNSE = left(firstNSE, 1)

Set statementRange = .Bookmarks("tca_fStatement").Range
statementRange.Paste
.Tables(1).ConvertToText
15 .Bookmarks.Add name:="tca_fStatement", Range:=statementRange
statementRange.Borders.OutsideLineStyle = wdLineStyleSingle

' trim hard returns at end of statement
Dim i, n As Integer
Dim retchr As String
20 retchr = Chr\$(13)

With statementRange
n = 0
i = .Words.Count

While .Words(i).Text = retchr And i > 1
25 i = i - 1
If .Words(i).Text = retchr Then
n = n + 1
End If
Wend

30 If n > 0 Then
.Words(.Words.Count - n + 1).Delete Count:=n
End If
End With

rnumber = .Tables(2).Rows.Count * Rnd + 0.5
35 .Tables(2).Cell(Row:=rnumber, Column:=1).Range.Copy

```
secondNSE = .Tables(2).Cell(Row:=rnumber, Column:=2).Range.Text
secondNSE = left(secondNSE, 1)
```

```
Set statementRange = .Bookmarks("tca_sStatement").Range
statementRange.Paste
5 .Tables(1).ConvertToText
.Bookmarks.Add name:="tca_sStatement", Range:=statementRange
statementRange.Borders.OutsideLineStyle = wdLineStyleSingle
```

```
' trim hard returns at end of statement
```

```
With statementRange
```

```
10 n = 0
```

```
i = .Words.Count
```

```
While .Words(i).Text = retchr And i > 1
```

```
i = i - 1
```

```
If .Words(i).Text = retchr Then
```

```
15 n = n + 1
```

```
End If
```

```
Wend
```

```
If n > 0 Then
```

```
.Words(.Words.Count - n + 1).Delete Count:=n
```

```
20 End If
```

```
End With
```

```
Dim key As String
```

```
Dim keyChr As String
```

```
If firstNSE = "N" And secondNSE = "N" Then
```

```
25 key = "E"
```

```
ElseIf firstNSE = "S" And secondNSE = "S" Then
```

```
key = "C or E"
```

```
ElseIf firstNSE = "E" And secondNSE = "E" Then
```

```
key = "D"
```

```
30 ElseIf firstNSE = "N" And secondNSE = "S" Then
```

```
key = "E"
```

```
ElseIf firstNSE = "E" And secondNSE = "S" Then
```

```
key = "A"
```

```
ElseIf firstNSE = "S" And secondNSE = "E" Then
```

```
35 key = "B"
```

```
ElseIf firstNSE = "N" And secondNSE = "E" Then
```

```
key = "B"
```

```
ElseIf firstNSE = "E" And secondNSE = "N" Then
```

```
key = "A"
```

End If

keyChr = left(.Bookmarks("key").Range.Text, 1)

If keyChr = "A" Or keyChr = "1" Then

key = "A"

ElseIf keyChr = "B" Or keyChr = "2" Then

key = "B"

ElseIf keyChr = "C" Or keyChr = "3" Then

key = "C"

ElseIf keyChr = "D" Or keyChr = "4" Then

key = "D"

ElseIf keyChr = "E" Or keyChr = "5" Then

key = "E"

End If

Dim keyRange As Range

Set keyRange = .Bookmarks("tca_Key").Range

If key = "" Then

keyRange.InsertBefore Text:="TCA cannot determine the key"

Else

keyRange.InsertBefore Text:="Key is " & key

End If

udtClone.key = key

End With

End Sub

```

' Family.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1 'True
5  END
  Attribute VB_Name = "Family"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = True
  Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
  Option Explicit

  ' current version of data produced by this class
  Const mintVERSIONSTAMP As Integer = 1

  ' enable i/o
15  Private mudtFile As File

  ' the .mdf file name of this family
  Private mstrFamilyFN As String

  ' the program that owns this family
  Private mudtProgram As Program

20  ' the item type
  Private mudtItemType As ItemType

  ' close/medium far classification
  Private mudtProximity As Proximity

  ' generic/non-generic classification
25  Private mblnGeneric As Boolean

  ' accession number, if this family is based on a locked item
  Private mstrAccNum As String

  ' the active model
  Private mudtActiveModel As Model

30  ' collection of Models
  Private mudtCModels As CModels

  ' the collection of accepted clones
  Private mudtCClones As CClones

```

```

' is dirty?
Private mblnIsDirty As Boolean

Public Enum Program
    prGRE = 0
5    prGMAT = 1
    prSAT = 2
    prMR = 3
End Enum

Public Enum ItemType
10    ptStandardMC = 0
    ptQuantComp = 1
    ptDataSuff = 2
End Enum

Public Enum Proximity
15    prNear = 0
    prMedium = 1
    prFar = 2
End Enum

Private Enum FamilyRecordLayout
20    frLocalDataIndex = 1 ' long (takes 4 bytes)
    frCloneIndex = 5 ' long
    frLocalData = 51
    frClones = 201 ' variable length
End Enum

25 Private Sub Class_Initialize()

    Set mudtCModels = New CModels
    Set mudtCClones = New CClones
    mblnIsDirty = False

End Sub

30 Public Property Get FileName() As String

    FileName = mstrFamilyFN

End Property

Public Property Let FileName(ByVal strNewValue As String)

```


mstrFamilyFN = left(strNewValue, Len(strNewValue) - 4) & ".mdf"

End Property

Public Property Get Program() As Program

5 Program = mudtProgram

End Property

Public Property Let Program(ByVal udtNewValue As Program)

10 mudtProgram = udtNewValue

End Property

Public Property Get ItemType() As ItemType

15 ItemType = mudtItemType

End Property

Public Property Let ItemType(ByVal udtNewValue As ItemType)

 mudtItemType = udtNewValue

End Property

20 Public Property Get Proximity() As Proximity

 Proximity = mudtProximity

End Property

Public Property Let Proximity(ByVal udtNewValue As Proximity)

 mudtProximity = udtNewValue

25 End Property

Public Property Get Generic() As Boolean

 Generic = mblnGeneric

End Property

Public Property Let Generic(ByVal blnNewValue As Boolean)

 mblnGeneric = blnNewValue

End Property

Public Property Get AccNum() As String

 AccNum = mstrAccNum

End Property

Public Property Let AccNum(ByVal strNewValue As String)

 mstrAccNum = strNewValue

End Property

Public Property Get ActiveModel() As Model

 Set ActiveModel = mudtActiveModel

End Property

Public Property Let ActiveModel(ByVal udtModel As Model)

 Set mudtActiveModel = udtModel

End Property

Public Property Get Models() As CModels

 Set Models = mudtCModels

End Property

Public Property Get Clones() As CClones

 Set Clones = mudtCClones

End Property

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

 mblnIsDirty = blnNewValue

```

End Property

Private Property Get IsDirty() As Boolean

    If mudtCClones.IsDirty Or mblnIsDirty Then
5      IsDirty = True
    Else
        IsDirty = False
    End If

10 End Property

Public Sub CloseAllCloneDocs()

    Dim udtClone As Clone

    For Each udtClone In mudtCClones
15      udtClone.CloseDoc
    Next udtClone

End Sub

Public Sub ReadFamily()

20      Dim udtWAPI As New Win32API

    If udtWAPI.FileExists(mstrFamilyFN) Then
        Set mudtFile = New File
        mudtFile.FileName = mstrFamilyFN
25      Call mudtFile.ReadFile(Me, frLocalDataIndex, frCloneIndex)
        Set mudtFile = Nothing
        Call mudtCClones.ReadCollection(mstrFamilyFN, frCloneIndex, READ_UNTIL_EOF)
    End If

30 End Sub

Public Sub ReadObjects()

    Dim vField As Variant

    Call mudtFile.ReadField(vField) ' returns the version stamp
35      Call mudtFile.ReadField(vField)
    Program = vField
    Call mudtFile.ReadField(vField)

```

```
ItemType = vField
Call mudtFile.ReadField(vField)
Generic = vField
Call mudtFile.ReadField(vField)
5 Proximity = vField
Call mudtFile.ReadField(vField)
AccNum = vField
```

```
End Sub
```

```
10 Public Sub WriteFamily()
```

```
Dim udtPB As New Progress
```

```
If IsDirty Then
```

```
Set mudtFile = New File
```

```
15 mudtFile.FileName = mstrFamilyFN
```

```
Call udtPB.Init(2, "Saving family...")
```

```
Call mudtFile.WriteFile(Me, True, frLocalDataIndex, frLocalData)
```

```
udtPB.Advance
```

```
Set mudtFile = Nothing
```

```
20 Call mudtCClones.WriteCollection(mstrFamilyFN, frCloneIndex, frClones)
```

```
udtPB.Advance
```

```
End If
```

```
IsDirty = False
```

```
25 End Sub
```

```
Public Sub WriteObjects()
```

```
Call mudtFile.WriteField(mintVERSIONSTAMP)
```

```
Call mudtFile.WriteField(Program)
```

```
30 Call mudtFile.WriteField(ItemType)
```

```
Call mudtFile.WriteField(Generic)
```

```
Call mudtFile.WriteField(Proximity)
```

```
Call mudtFile.WriteField(AccNum)
```

```
End Sub
```

```

' File.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = 0 'False
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "File"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    ' Path and name of the file to open
    Private m_sFileName As String

    ' File number opened
    Private m_iFileNumber As Integer

20    ' passed in by ReadFile
    Private mlngEndPos As Long

    ' Error constants
    Enum FileError
        fileOpenError = vbObjectError + 512 + 2
25        fileEOFError = vbObjectError + 512 + 3
        fileReadError = vbObjectError + 512 + 4
        fileWriteError = vbObjectError + 512 + 5
        fileStopReadingError = vbObjectError + 512 + 6
    End Enum

30    Property Get FileName() As String
    Attribute FileName.VB_Description = "Name of the file to contain the task information."

        FileName = m_sFileName

    End Property

35    Property Let FileName(ByVal sFileName As String)

        ' Should validate valid path here

```

```
m_sFileName = sFileName
```

```
End Property
```

```
' Reads all objects from a file into the defined object
```

```
5 ' Parameters:
```

```
Public Sub ReadFile(obj As Object, Optional ByVal lngStartIndex As Long = 0, _  
    Optional ByVal lngEndIndex As Long = 0)
```

```
    Dim lngStartPos As Long
```

```
10 ' Enable error handling  
    On Error Resume Next
```

```
    ' Get the file number  
    m_iFileNumber = FreeFile
```

```
15 ' Open the file and trap any errors  
    Open m_sFileName For Binary Access Read As #m_iFileNumber
```

```
    Select Case err.Number
```

```
20 Case 0 ' No error  
    If lngEndIndex > 0 Then  
        Seek m_iFileNumber, lngEndIndex  
        Get #m_iFileNumber, , mlngEndPos  
    Else  
        mlngEndPos = 0  
    End If  
    If lngStartIndex > 0 Then  
        Seek m_iFileNumber, lngStartIndex  
        Get #m_iFileNumber, , lngStartPos  
        Seek m_iFileNumber, lngStartPos  
    End If  
    obj.ReadObjects ' Get the data
```

```
35 Case 53 ' File not found  
    ' Do nothing
```

```
Case Else  
    ' Turn off error handling here  
40 On Error GoTo 0
```

```
    ' Pass the error out  
    err.Raise fileOpenError, "CFile::ReadFile", "Error opening file."
```

```

End Select

' Close the file

5   Close #m_iFileNumber

End Sub

' Reads a field from the file
' Parameters:
10  ' vField    field read from the file

Public Sub ReadField(vField As Variant)
    ' Set the error handler
    On Error GoTo ERR_HANDLER

15   Get #m_iFileNumber, , vField

    If EOF(m_iFileNumber) Then
        ' Reached end of file
        err.Raise fileEOFError
20   End If

    If mlngEndPos > 0 Then
        If mlngEndPos < Seek(m_iFileNumber) Then
            err.Raise fileStopReadingError
25   End If
        End If

Exit Sub

ERR_HANDLER:
30   ' Pass the error out
    Select Case err.Number

        Case fileEOFError
            Call err.Raise(err.Number, "File::ReadField", "EOF")
35   Case fileStopReadingError
            Call err.Raise(err.Number, "File::ReadField", "Stop!")
        Case Else
            Call err.Raise(fileReadError, "File::ReadField", err.Description)

40   End Select

```

End Sub

' Writes all objects to the file.

' Parameters:

' obj Object

5 Public Function WriteFile(obj As Object, _
 Optional ByVal blnKillOldFile As Boolean = False, _
 Optional ByVal lngIndexPos As Long = 0, _
 Optional ByVal lngSeekPos As Long = 1) As Long

' Enable error handling

10 On Error Resume Next

If blnKillOldFile Then ' assume new file, otherwise append

 Kill m_sFileName ' Kill the existing file

 err.Clear

End If

15 ' Get the file number

m_iFileNumber = FreeFile

' Open the file and trap any errors

Open m_sFileName For Binary As #m_iFileNumber

20 ' write the starting file position, if lngIndexPos > 0

If lngIndexPos > 0 Then

 Seek m_iFileNumber, lngIndexPos

 Put #m_iFileNumber, , lngSeekPos

End If

25 ' seek to starting position

Seek m_iFileNumber, lngSeekPos

Select Case err.Number

30 Case 0 ' No error

 ' Write the data

 obj.WriteObjects

Case Else

 ' Turn off error handling here

35 On Error GoTo 0

 ' Pass the error out

err.Raise fileOpenError, "CFile::WriteFile", _

 "Error opening file: " & err.Description

End Select

' return current position

5 WriteFile = Seek(m_iFileNumber)

' Close the file

Close #m_iFileNumber

10 End Function

' Write a field to the file

' Parameters:

' vField field to write to the file

Public Sub WriteField(ByVal vField As Variant)

15 ' Set the error handler

On Error GoTo ERR_HANDLER

Put #m_iFileNumber, , vField

Exit Sub

ERR_HANDLER:

20 err.Raise fileWriteError, "CFile::WriteField", _

"Write Error: " & err.Description

End Sub

```

' FileFind.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "FileFind"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

' used for finding files that fit a mask

Private Type FILETIME
15     dwLowDateTime As Long
     dwHighDateTime As Long
End Type

Private Const MAX_PATH = 260

Private Type WIN32_FIND_DATA
20     dwFileAttributes As Long
     ftCreationTime As FILETIME
     ftLastAccessTime As FILETIME
     ftLastWriteTime As FILETIME
     nFileSizeHigh As Long
25     nFileSizeLow As Long
     dwReserved0 As Long
     dwReserved1 As Long
     cFileName As String * MAX_PATH
     cAlternate As String * 14
30 End Type

Private Const INVALID_HANDLE_VALUE = -1

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
    (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long

35 Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" _
    (ByVal hFileName As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long

```

```
Private Declare Function GetCurrentDirectory Lib "kernel32" _
    Alias "GetCurrentDirectoryA" (ByVal nBufferLength As Long, _
    ByVal lpBuffer As String) As Long
```

```
5 ' returns true if strFN exists
```

```
Public Function Exists(ByVal strFN) As Boolean
```

```
    Dim lngHandle As Long
```

```
    Dim w32FindData As WIN32_FIND_DATA
```

```
    lngHandle = FindFirstFile(strFN, w32FindData)
```

```
10
```

```
    If lngHandle = INVALID_HANDLE_VALUE Then
```

```
        Exists = False
```

```
    Else
```

```
        Exists = True
```

```
15
```

```
        Call FindClose(lngHandle)
```

```
    End If
```

```
End Function
```

```
' returns a collection of file names that satisfy strMask. The path seems to  
' disappear from the returned file names.
```

```
20
```

```
Public Function FindAll(ByVal strMask As String) As Collection
```

```
    Dim lngHandle As Long
```

```
    Dim lngRet As Long
```

```
    Dim w32FindData As WIN32_FIND_DATA
```

```
    Dim strFN As String
```

```
25
```

```
    Dim varI As Variant
```

```
    Dim colFNs As New Collection
```

```
    lngHandle = FindFirstFile(strMask, w32FindData)
```

```
    If lngHandle = INVALID_HANDLE_VALUE Then
```

```
30
```

```
        Exit Function
```

```
    End If
```

```
    Do
```

```
        varI = InStr(1, w32FindData.cFileName, Chr(0)) ' trim off the nulls
```

```
35
```

```
        strFN = left(w32FindData.cFileName, varI - 1)
```

```
        Call colFNs.Add(strFN) ' add to the collection
```

```
    Loop Until FindNextFile(lngHandle, w32FindData) = 0
```

Set FindAll = colFNs

End Function

5 ' returns the current directory

Public Function CurrentDirectory() As String

Dim strBuf As String

Dim lngRet As Long

Dim varI As Variant

10

strBuf = Space(300)

lngRet = GetCurrentDirectory(300, strBuf)

varI = InStr(1, strBuf, Chr(0)) ' trim off the nulls

CurrentDirectory = left(strBuf, varI - 1)

15

End Function

```

' GMATDifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "GMATDifficultyEstimate"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Implements DifficultyEstimate

15 Private mudtDE As DifficultyEstimate

' these go into the GMAT model
Private mudtDomain As Domain
Private mstrKey As String
Private mudtNature As Nature
20 Private mudtItemType As ItemType
Private mintTDDiffEst As Integer

Private Sub Class_Initialize()

    Set mudtDE = New DifficultyEstimate

25 End Sub

Private Sub Class_Terminate()

    Set mudtDE = Nothing

End Sub

Public Property Get DifficultyEstimate_IsDirty() As Boolean

30     DifficultyEstimate_IsDirty = mudtDE.IsDirty

End Property

```

Public Property Let DifficultyEstimate_IsDirty(ByVal blnNewValue As Boolean)

 mudtDE.IsDirty = blnNewValue

End Property

5 Public Property Let Domain(ByVal udtNewValue As Domain)

 mudtDomain = udtNewValue

End Property

Public Property Let Nature(ByVal udtNewValue As Nature)

10 mudtNature = udtNewValue

End Property

Public Property Let Key(ByVal strNewValue As String)

 mstrKey = strNewValue

End Property

15 Public Property Let ItemType(ByVal udtNewValue As ItemType)

 mudtItemType = udtNewValue

End Property

Public Property Let TDDiffEst(ByVal intNewValue As Integer)

 mintTDDiffEst = intNewValue

20 End Property

Public Function DifficultyEstimate_ComputeDifficulty() As Double

 Dim dblDiff As Double

 dblDiff = -2.3289902

25

 ' add coeff for domain

 If mudtDomain = doAlgebra Then

 dblDiff = dblDiff + 0.2341578

```
ElseIf mudtDomain = doGeometry Then
    dblDiff = dblDiff + 0.3749013
End If
```

```
5      ' add coeff for real
      If mudtNature = naReal Then
          dblDiff = dblDiff + 0.3285613
      End If
```

```
10     ' add coeff for td difficulty estimate
      dblDiff = dblDiff + ((6 - mintTDDiffEst) * 0.7024191)
```

```
      ' add coeff for key
      If mudtItemType = ptDataSuff Then
15         If mstrKey = "A" Or mstrKey = "B" Then
            dblDiff = dblDiff + 0.7334054
        End If
      End If
```

```
20     DifficultyEstimate_ComputeDifficulty = dblDiff
```

```
End Function
```

```
' returns a copy of this object
Public Function DifficultyEstimate_Copy() As DifficultyEstimate
```

```
25     Dim udtGmatDE As New GMATDifficultyEstimate
```

```
     Set DifficultyEstimate_Copy = udtGmatDE
```

```
End Function
```

```
Public Sub DifficultyEstimate_ReadObjectData(udtFile As File)
```

```
30     Dim vField As Variant
```

```
     Call udtFile.ReadField(vField) ' reads the version stamp
```

```
End Sub
```

```
35 Public Sub DifficultyEstimate_WriteObjectData(udtFile As File)
```

```
     Call udtFile.WriteField(mintVERSIONSTAMP)
```

```
     mudtDE.IsDirty = False
```

End Sub


```

' GREDifficultyEstimate.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "GREDifficultyEstimate"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Implements DifficultyEstimate

15  Private mudtDE As DifficultyEstimate

' these go into the GRE model
Private mudtDomain As Domain
Private mudtComputation As GREComputation
Private mudtCognition As GRECognition
20  Private mudtConcept As GREConcept
Private mstrKey As String
Private mudtNature As Nature
Private mudtItemType As ItemType

Public Enum GREComputation
25  grIntegers = 0
    grDecimalsFractions = 1
    grRadicals = 2
    grNone = 3
End Enum

30  Public Enum GRECognition
    grProcedural = 0
    grConceptual = 1
    grHigherOrderThinking = 2
End Enum

35  Public Enum GREConcept
    grProbability = 0
    grPercentofPercent = 1

```

```
    grPercentChange = 2
    grLinearInequality = 3
    grNoneOfThese = 4
End Enum
```

```
5 Private Sub Class_Initialize()
```

```
    Set mudtDE = New DifficultyEstimate
```

```
End Sub
```

```
Private Sub Class_Terminate()
```

```
10    Set mudtDE = Nothing
```

```
End Sub
```

```
Public Property Get DifficultyEstimate_IsDirty() As Boolean
```

```
    DifficultyEstimate_IsDirty = mudtDE.IsDirty
```

```
End Property
```

```
15 Public Property Let DifficultyEstimate_IsDirty(ByVal blnNewValue As Boolean)
```

```
    mudtDE.IsDirty = blnNewValue
```

```
End Property
```

```
Public Property Let Domain(ByVal udtNewValue As Domain)
```

```
20    mudtDomain = udtNewValue
```

```
End Property
```

```
Public Property Get Computation() As GREComputation
```

```
    Computation = mudtComputation
```

```
25 End Property
```

```
Public Property Let Computation(ByVal udtNewValue As GREComputation)
```

```
    If mudtComputation <> udtNewValue Then
        mudtComputation = udtNewValue
```

```
        mudtDE.IsDirty = True
    End If
```

```
End Property
```

```
5 Public Property Get Cognition() As GRECognition
```

```
    Cognition = mudtCognition
```

```
End Property
```

```
Public Property Let Cognition(ByVal udtNewValue As GRECognition)
```

```
    If mudtCognition <> udtNewValue Then
10         mudtCognition = udtNewValue
        mudtDE.IsDirty = True
    End If
```

```
End Property
```

```
Public Property Get Concept() As GREConcept
```

```
15 Concept = mudtConcept
```

```
End Property
```

```
Public Property Let Concept(ByVal udtNewValue As GREConcept)
```

```
    If mudtConcept <> udtNewValue Then
20         mudtConcept = udtNewValue
        mudtDE.IsDirty = True
    End If
```

```
End Property
```

```
Public Property Get Nature() As Nature
```

```
    Nature = mudtNature
```

```
25 End Property
```

```
Public Property Let Nature(ByVal udtNewValue As Nature)
```

```
    mudtNature = udtNewValue
```

End Property

Public Property Get Key() As String

Key = mstrKey

End Property

5 Public Property Let Key(ByVal strNewValue As String)

If mstrKey <> strNewValue Then

mstrKey = strNewValue

mudtDE.IsDirty = True

End If

10 End Property

Public Property Get ItemType() As ItemType

ItemType = mudtItemType

End Property

Public Property Let ItemType(ByVal udtNewValue As ItemType)

15 mudtItemType = udtNewValue

End Property

Public Function DifficultyEstimate_ComputeDifficulty() As Double

Dim dblDiff As Double

20 dblDiff = 0.3296816

' add coeff for domain

If mudtDomain = doAlgebra Then

dblDiff = dblDiff + 0.2464302

25 ElseIf mudtDomain = doDataAnalysis Then

dblDiff = dblDiff - 0.3944198

End If

' add coeff for computation

30 If mudtComputation = grIntegers Then

dblDiff = dblDiff - 0.8563799

```
ElseIf mudtComputation = grDecimalsFractions Then
    dblDiff = dblDiff - 0.5181709
End If
```

```
5 ' add coeff for cognition
If mudtCognition = grProcedural Then
    dblDiff = dblDiff - 0.6621277
    If mudtNature = naReal Then ' add coeff for procedural and real
        dblDiff = dblDiff - 0.8781659
10 End If
ElseIf mudtCognition = grHigherOrderThinking Then
    dblDiff = dblDiff + 0.7253093
End If
```

```
15 ' add coeff for concept
Select Case mudtConcept
    Case grLinearInequality
        dblDiff = dblDiff - 0.5881492
    Case grNoneOfThese
20 ' do nothing
    Case Else
        dblDiff = dblDiff + 0.5835095
End Select
```

```
25 ' add coeff for key
If mudtItemType = ptQuantComp Then
    If mstrKey = "A" Or mstrKey = "B" Or mstrKey = "C" Then
        dblDiff = dblDiff - 0.531099
    End If
30 End If
```

```
DifficultyEstimate_ComputeDifficulty = dblDiff
```

```
End Function
```

```
35 ' returns a copy of this object
Public Function DifficultyEstimate_Copy() As DifficultyEstimate
```

```
    Dim udtGreDE As New GREDifficultyEstimate
```

```
    udtGreDE.Computation = Computation
    udtGreDE.Cognition = Cognition
40 udtGreDE.Concept = Concept
```

```
Set DifficultyEstimate_Copy = udtGreDE
```

End Function

Public Sub DifficultyEstimate_ReadObjectData(udtFile As File)

Dim vField As Variant

5 Call udtFile.ReadField(vField) ' reads the version stamp

Call udtFile.ReadField(vField)
Computation = vField

10 Call udtFile.ReadField(vField)
Cognition = vField

Call udtFile.ReadField(vField)
Concept = vField

End Sub

15 Public Sub DifficultyEstimate_WriteObjectData(udtFile As File)

Call udtFile.WriteField(mintVERSIONSTAMP)
Call udtFile.WriteField(Computation)
Call udtFile.WriteField(Cognition)
20 Call udtFile.WriteField(Concept)

mutdDE.IsDirty = False

End Sub

25 ' IniFile.cls
VERSION 1.0 CLASS
BEGIN

MultiUse = -1 'True

END

30 Attribute VB_Name = "IniFile"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False

35 ' this class handles all ini file reads and writes via kernel32

Option Explicit

' the following declares are needed to get and put to .ini files

```
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias _  
    "GetPrivateProfileSectionA" (ByVal lpAppName As String, _  
5     ByVal lpReturnedString As String, ByVal nSize As Long, _  
        ByVal lpFileName As String) As Long
```

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _  
    "GetPrivateProfileStringA" (ByVal lpApplicationName As String, _  
10     ByVal lpKeyName As Any, ByVal lpDefault As String, _  
        ByVal lpReturnedString As String, ByVal nSize As Long, _  
        ByVal lpFileName As String) As Long
```

```
Private Declare Function WritePrivateProfileSection Lib "kernel32" Alias _  
15     "WritePrivateProfileSectionA" (ByVal lpAppName As String, _  
        ByVal lpString As String, ByVal lpFileName As String) As Long
```

```
Private Declare Function WritePrivateProfileString Lib "kernel32" Alias _  
    "WritePrivateProfileStringA" (ByVal lpApplicationName As String, _  
20     ByVal lpKeyName As Any, ByVal lpString As Any, ByVal lpFileName As String) _  
        As Long
```

' contains file name of ini
Private mstrFN As String

' holds collection of keys created by Get ProfileSection method
25 Private mcolKeys As Collection

' holds collection of values created by Get ProfileSection method
Private mcolValues As Collection

Private Sub Class_Initialize()

```
30     Set mcolKeys = New Collection  
     Set mcolValues = New Collection
```

End Sub

' sets the ini path + file name
Public Property Let FN(ByVal strFN As String)

```
35     mstrFN = strFN
```

End Property

' returns the ini path + file name
Public Property Get FN() As String

FN = mstrFN

5 End Property

'gets all of the keys and values in a section
Public Sub GetProfileSection(ByVal strSectionName As String)

Dim strRet As String
strRet = Space(5000)

10 If GetPrivateProfileSection(strSectionName, strRet, 5000, mstrFN) = 0 Then
Call MsgBox("Ini file call unsuccessful", vbExclamation, "Error")
End If

15 Dim lngStart As Long
Dim lngEnd As Long
Dim str1 As String
Dim str2 As String
Dim varT As Variant
Dim strT As String

20 ' parse the key and variable names out of strRet, add to the collections

For lngStart = 1 To Len(strRet)
str1 = Mid(strRet, lngStart, 1)
If str1 <> Chr(0) Then
For lngEnd = lngStart + 1 To Len(strRet)
25 str2 = Mid(strRet, lngEnd, 1)
Select Case str2
Case "="
strT = Mid(strRet, lngStart, lngEnd - lngStart)
Call mcolKeys.Add(strT)
Exit For
30 Case Chr(0)
strT = Mid(strRet, lngStart, lngEnd - lngStart)
Call mcolValues.Add(strT)
Exit For
35 End Select
Next lngEnd
lngStart = lngEnd
End If
Next lngStart

40

End Sub

' called after LoadProfileSection.

' sets strKey and strValue to the the KeyValue pairs if one exists

' at this index.

5 ' returns TRUE if the index exists, FALSE if it doesn't.

Public Function GetKeyValuePair(strKey As String, strValue As String, _
ByVal intIndex As Integer) As Boolean

If intIndex <= mcolKeys.Count Then

strKey = mcolKeys.Item(intIndex)

10 strValue = mcolValues.Item(intIndex)

GetKeyValuePair = True

Else

strKey = ""

strValue = ""

15 GetKeyValuePair = False

End If

End Function

' init before loading key/value pairs

20 Public Function InitializeKeyValuePairs()

Set mcolKeys = Nothing

Set mcolValues = Nothing

Set mcolKeys = New Collection

Set mcolValues = New Collection

25 End Function

Public Sub SetKeyValuePair(ByVal strKey As String, ByVal strValue As String)

Call mcolKeys.Add(strKey)

Call mcolValues.Add(strValue)

End Sub

30 Public Sub WriteProfileSection(ByVal strSectionName As String)

Dim strSection As String

Dim varKey As Variant

Dim varValue As Variant

Dim intI As Integer

```

For Each varKey In mcolKeys
    intI = intI + 1
    varValue = mcolValues.Item(intI)
    strSection = strSection & varKey & "=" & varValue & Chr(0)
5 Next varKey

If WritePrivateProfileSection(strSectionName, strSection, mstrFN) = 0 Then
    Call MsgBox("Ini file write section call unsuccessful", _
        vbExclamation, "Error")
10 End If

End Sub

' returns the number of keys currently in the key/value collections
Public Property Get NumKeys() As Integer

15     NumKeys = mcolKeys.Count

End Property

'gets a value
Public Function GetProfileString(ByVal strSectionName As String, _
20     ByVal strKeyName As String) As String

    Dim strRet As String
    strRet = Space(5000)

    Call GetPrivateProfileString(strSectionName, strKeyName, "Not Found", _
        strRet, 5000, mstrFN)
25     GetProfileString = TrimAtFirstNull(strRet)

End Function

'sets a value
Public Sub WriteProfileString(ByVal strSectionName As String, _
    ByVal strKeyName As String, ByVal strKeyValue As String)

30     Call WritePrivateProfileString(strSectionName, strKeyName, strKeyValue, _
        mstrFN)

End Sub

```

```
' LockedItem.cls
VERSION 1.0 CLASS
BEGIN
```

```
MultiUse = -1 'True
```

```
END
```

```
Attribute VB_Name = "LockedItem"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = True
```

```
Attribute VB_PredeclaredId = False
```

```
Attribute VB_Exposed = False
```

```
Option Explicit
```

```
Private mstrLockedFN As String
```

```
Private mudtWord As MSWord
```

```
Private mdocLockedItem As Document
```

```
Private mudtItemType As ItemType
```

```
Private mudtDeliveryMode As DeliveryMode
```

```
Public Enum DeliveryMode
```

```
dmCBT = 0
```

```
dmPPT = 1
```

```
End Enum
```

```
Public Property Let LockedItemFileName(ByVal strNewValue As String)
```

```
mstrLockedFN = strNewValue
```

```
End Property
```

```
Public Property Let WordInstance(ByVal udtNewValue As MSWord)
```

```
Set mudtWord = udtNewValue
```

```
End Property
```

```
Public Property Get DeliveryMode() As DeliveryMode
```

```
DeliveryMode = mudtDeliveryMode
```

```
End Property
```

```
Public Property Get ItemType() As ItemType
```

```
    ItemType = mudtItemType
```

```
End Property
```

```
Public Function OpenLockedItemDoc() As Boolean
```

```
5    Dim udtProgress As New Progress
```

```
    Call udtProgress.Init(2, "Opening locked item...")  
    udtProgress.Advance
```

```
    Set mdocLockedItem = mudtWord.WordApp.Documents.Open(mstrLockedFN)
```

```
10    If mdocLockedItem.ProtectionType <> wdNoProtection Then  
        Call mdocLockedItem.Unprotect("ItemEdit")  
    End If
```

```
    OpenLockedItemDoc = AnalyzeLockedItem
```

```
    udtProgress.Advance
```

```
End Function
```

```
15 Public Sub CloseLockedItemDoc()
```

```
    mdocLockedItem.Close
```

```
    Clipboard.Clear
```

```
End Sub
```

```
Private Function AnalyzeLockedItem() As Boolean
```

```
20    ' true if document is successfully analyzed  
    AnalyzeLockedItem = True
```

```
    If mdocLockedItem.Tables.Count = 1 Then ' QC item  
        mudtItemType = ptQuantComp
```

```
25    If mdocLockedItem.Bookmarks.Count = 3 Then  
        mudtDeliveryMode = dmPPT
```

```
    Else
```

```
        mudtDeliveryMode = dmCBT
```

```
    End If
```

```
ElseIf mdocLockedItem.ListParagraphs.Count = 2 Then ' DS
    mudtItemType = ptDataSuff
    mudtDeliveryMode = dmCBT
```

```
ElseIf mdocLockedItem.ListParagraphs.Count = 5 Then ' SMC
    mudtItemType = ptStandardMC
    mudtDeliveryMode = dmCBT
```

```
ElseIf mdocLockedItem.Bookmarks.Exists("prop_key") = True Then 'SMC
    mudtItemType = ptStandardMC
    mudtDeliveryMode = dmPPT
```

```
Else
    AnalyzeLockedItem = False
End If
```

```
End Function
```

```
Public Sub ConvertCBTSMCItem()
```

```
    Dim udtProgress As New Progress
```

```
    Call udtProgress.Init(2, "Converting SMC CBT locked item...")
```

```
    Dim tcaDoc As Document
    Set tcaDoc = mudtWord.WordApp.ActiveDocument
```

```
    Dim stemRange As Range
    Set stemRange = mdocLockedItem.Content
    stemRange.Find.Style = "Heading 2"
    stemRange.Find.Execute FindText:="Stem"
    stemRange.Start = stemRange.Start + 5
```

```
    Dim respRange As Range
    Set respRange = mdocLockedItem.Content
    respRange.Find.Style = "Heading 2"
    respRange.Find.Execute FindText:="Response"
```

```
    stemRange.End = respRange.Start - 1
    stemRange.Copy
```

```
    Dim destRange As Range
    Set destRange = tcaDoc.Bookmarks("stem1").Range
```

```
    With destRange
```

```

' .Borders.Enable = False
' .Words(1).Delete Count:=6
' .Collapse
' .Paste
5 ' .Style = wdStyleNormal
' .Borders.Enable = True
End With

' destRange.Borders.Enable = False
' destRange.Collapse
10 ' destRange.Delete
' destRange.Paste
' destRange.InsertParagraphAfter
' destRange.Style = wdStyleNormal
' destRange.Borders.Enable = True

15 With destRange.ParagraphFormat.Borders
' .Enable = True
' .DistanceFromTop = 1
' .DistanceFromLeft = 4
' .DistanceFromBottom = 1
20 ' .DistanceFromRight = 4
End With

If destRange.Borders.InsideLineStyle = True Then
' destRange.Borders.InsideLineStyle = wdLineStyleNone
End If

25 ' tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

Dim nextRange As Range
Dim Key As String
Dim abcde As String
abcde = "ABCDE"
30 Dim i As Integer
Dim n As Integer
n = 1

Dim udtIF As New IniFile
udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
35 Key = udtIF.GetProfileString("LockedItemData", "Key")

udtProgress.Advance

Dim tabchr As String

```

tabchr = Chr\$(9)

For i = 1 To 5

Set respRange = mdocLockedItem.ListParagraphs(i).Range
respRange.Copy

5 If Key = Mid(abcde, i, 1) Then
Set destRange = tcaDoc.Bookmarks("Key").Range
Else
Set destRange = tcaDoc.Bookmarks("resp" & Format(n)).Range
n = n + 1
10 End If

With destRange
.Borders.Enable = False
.Words(1).Delete
.Collapse
15 .Paste
.Style = wdStyleNormal
.Borders.Enable = True

If .Words(1).Text = tabchr Then
.Words(1).Delete
20 End If

.Words(destRange.Words.Count).Delete
End With

Next

udtProgress.Advance

25 End Sub

Public Sub ConvertPPTSMCItem()

Dim udtProgress As New Progress

Call udtProgress.Init(2, "Converting SMC PPT locked item...")

30 Dim tcaDoc As Document
Set tcaDoc = mdtWord.WordApp.ActiveDocument

Dim stemStart As Long

```
Dim destRange As Range
Set destRange = tcaDoc.Bookmarks("stem1").Range
stemStart = destRange.Start
```

```
Dim stemRange As Range
5 ' Set stemRange = mdocLockedItem.Bookmarks("itemnum").Range
' stemRange.Start = stemRange.Start + 1
Set stemRange = mdocLockedItem.Content
stemRange.Find.Style = "PPTStimulus"
```

```
10 If stemRange.Find.Execute Then
    stemRange.Copy
    destRange.Paste
    destRange.Collapse Direction:=wdCollapseEnd
End If
```

```
15 Set stemRange = mdocLockedItem.Content
stemRange.Find.Style = "PPTStem"
stemRange.Find.Execute
stemRange.Copy
destRange.Paste
destRange.Style = wdStyleNormal
```

```
20 destRange.Start = stemStart
destRange.Borders.Enable = True
```

```
25 ' With destRange.ParagraphFormat.Borders
' .Enable = True
' .DistanceFromTop = 1
' .DistanceFromLeft = 4
' .DistanceFromBottom = 1
' .DistanceFromRight = 4
' End With
```

```
30 If destRange.Borders.InsideLineStyle = True Then
    destRange.Borders.InsideLineStyle = wdLineStyleNone
End If
```

```
tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange
```

```
35 Dim nextRange As Range
Dim respRange As Range
Dim Key As String
Dim abcde As String
abcde = "ABCDE"
```



```
Dim i As Integer
Dim n As Integer
n = 1
```

```
Dim udtIF As New IniFile
5 udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
Key = udtIF.GetProfileString("LockedItemData", "Key")
```

```
udtProgress.Advance
```

```
For i = 1 To 5
```

```
10 Set respRange = mdocLockedItem.Content
respRange.Find.Style = "PPTOptions"
respRange.Find.Execute FindText:="(" & Mid(abcde, i, 1) & ")"
respRange.Start = respRange.Start + 4
```

```
Set nextRange = mdocLockedItem.Content
```

```
15 If i < 5 Then
nextRange.Find.Style = "PPTOptions"
nextRange.Find.Execute FindText:="(" & Mid(abcde, i + 1, 1) & ")"
Else
nextRange.Find.Style = "ItemLabel"
nextRange.Find.Execute FindText:="Scratch Pad"
20 End If
```

```
respRange.End = nextRange.Start - 1
respRange.Copy
```

```
25 If Key = Mid(abcde, i, 1) Then
Set destRange = tcaDoc.Bookmarks("Key").Range
Else
Set destRange = tcaDoc.Bookmarks("resp" & Format(n)).Range
n = n + 1
End If
```

```
30 destRange.Words(1).Delete
destRange.Collapse
destRange.Paste
```

```
Next
```

```
udtProgress.Advance
```

End Sub

Public Sub ConvertDSItem()

Dim udtProgress As New Progress

Call udtProgress.Init(2, "Converting DS CBT locked item...")

5 Dim tcaDoc As Document
Set tcaDoc = mudtWord.WordApp.ActiveDocument

Dim stemRange As Range
Set stemRange = mdocLockedItem.Content
stemRange.Find.Style = "Heading 2"
10 stemRange.Find.Execute FindText:="Stem"
stemRange.Start = stemRange.Start + 5

Dim respRange As Range
Set respRange = mdocLockedItem.Content
respRange.Find.Style = "DataSuffStatement"
15 respRange.Find.Execute

stemRange.End = respRange.Start - 1
stemRange.Copy

Dim destRange As Range
Set destRange = tcaDoc.Bookmarks("stem1").Range
20 destRange.Borders.Enable = False
destRange.Collapse
destRange.Paste
' destRange.Borders.Enable = True

With destRange.ParagraphFormat.Borders
25 .Enable = True
.DistanceFromTop = 1
.DistanceFromLeft = 4
.DistanceFromBottom = 1
.DistanceFromRight = 4
30 End With

If destRange.Borders.HasHorizontal = True Then
destRange.Borders(wdBorderHorizontal).LineStyle = wdLineStyleNone
End If

Dim Key As String

```
Dim udtIF As New IniFile
udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
Key = udtIF.GetProfileString("LockedItemData", "Key")
```

```
Set destRange = tcaDoc.Bookmarks("Key").Range
destRange.Words(1).Delete
destRange.InsertBefore Text:=Key
```

```
udtProgress.Advance
```

```
Dim i As Integer
```

```
For i = 1 To 2
```

```
Set respRange = mdocLockedItem.ListParagraphs(i).Range
respRange.Copy
```

```
Set destRange = tcaDoc.Tables(i).Cell(Row:=1, Column:=1).Range
destRange.Paste
destRange.Style = wdStyleNormal
```

```
Next
```

```
udtProgress.Advance
```

```
End Sub
```

```
Public Sub ConvertCBTQCItem()
```

```
Dim udtProgress As New Progress
```

```
Call udtProgress.Init(2, "Converting QC CBT locked item...")
```

```
Dim tcaDoc As Document
Set tcaDoc = mudtWord.WordApp.ActiveDocument
```

```
Dim stemRange As Range
Set stemRange = mdocLockedItem.Tables(1).Cell(Row:=1, Column:=1).Range
stemRange.Copy
```

```
Dim destRange As Range
Set destRange = tcaDoc.Bookmarks("stem1").Range
destRange.Borders.Enable = False
destRange.Words(2).Delete
destRange.Words(1).Delete
```

```

destRange.Collapse
destRange.Paste
tcaDoc.Tables(2).Rows.SetLeftIndent LeftIndent:=-0.6, RulerStyle:=wdAdjustNone
tcaDoc.Tables(2).ConvertToText Separator:=wdSeparateByTabs
5 destRange.Borders.Enable = True
tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange

```

```

Dim Key As String
Dim udtIF As New IniFile
10 udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
Key = udtIF.GetProfileString("LockedItemData", "Key")

```

```

Set destRange = tcaDoc.Bookmarks("Key").Range
destRange.Words(1).Delete
destRange.InsertBefore Text:=Key

```

```

udtProgress.Advance

```

```

15 Dim respRange As Range
Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=1).Range
respRange.Copy
Set destRange = tcaDoc.Bookmarks("columnA").Range
destRange.Collapse
20 destRange.Paste

```

```

Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=2).Range
respRange.Copy
Set destRange = tcaDoc.Bookmarks("columnB").Range
destRange.Collapse
25 destRange.Paste

```

```

udtProgress.Advance

```

```

End Sub

```

```

Public Sub ConvertPPTQCItem()

```

```

Dim udtProgress As New Progress

```

```

30 Call udtProgress.Init(2, "Converting QC PPT locked item...")

```

```

Dim tcaDoc As Document
Set tcaDoc = mdtWord.WordApp.ActiveDocument

```

```

Dim stemRange As Range

```

```
Set stemRange = mdocLockedItem.Tables(1).Cell(Row:=1, Column:=2).Range
stemRange.Copy
```

```
Dim destRange As Range
```

```
Set destRange = tcaDoc.Bookmarks("stem1").Range
```

```
destRange.Borders.Enable = False
```

```
destRange.Words(2).Delete
```

```
destRange.Words(1).Delete
```

```
destRange.Collapse
```

```
destRange.Paste
```

```
tcaDoc.Tables(2).Rows.SetLeftIndent LeftIndent:=-0.6, RulerStyle:=wdAdjustNone
```

```
tcaDoc.Tables(2).ConvertToText Separator:=wdSeparateByTabs
```

```
destRange.Borders.Enable = True
```

```
tcaDoc.Bookmarks.Add Name:="stem1", Range:=destRange
```

```
Dim Key As String
```

```
Dim udtIF As New IniFile
```

```
udtIF.FN = IN_DIRECTORY & ExtractFileNameNoExt(mstrLockedFN) & ".ini"
```

```
Key = udtIF.GetProfileString("LockedItemData", "Key")
```

```
Set destRange = tcaDoc.Bookmarks("Key").Range
```

```
destRange.Words(1).Delete
```

```
destRange.InsertBefore Text:=Key
```

```
udtProgress.Advance
```

```
Dim respRange As Range
```

```
Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=2).Range
```

```
respRange.Copy
```

```
Set destRange = tcaDoc.Bookmarks("columnA").Range
```

```
destRange.Collapse
```

```
destRange.Paste
```

```
Set respRange = mdocLockedItem.Tables(1).Cell(Row:=2, Column:=4).Range
```

```
respRange.Copy
```

```
Set destRange = tcaDoc.Bookmarks("columnB").Range
```

```
destRange.Collapse
```

```
destRange.Paste
```

```
udtProgress.Advance
```

```
End Sub
```



```

' Model.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "Model"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
    Attribute VB_Ext_KEY = "Top_Level" ,"No"
    Option Explicit

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

20    ' enable i/o
    Private mudtFile As File

    ' handle for Model
    Private mdocModel As Document

    ' the .doc file name of this model
25    Private mstrDocFN As String

    ' the .mdl file name of this model
    Private mstrConFN As String

    ' has this model produced variants that were accepted?
    Private mblnIsFrozen As Boolean

30    ' comments about this model
    Private mstrComments As String

    ' all of the variables for this model
    Private mudtCVariables As CVariables

    ' all of the constraints for this model
35    Private mudtCConstraints As CConstraints

```

```

' all of the clones generated by this model
Private mudtCClones As CClones

' the collection of checksums accepted by this model (these persist)
Private mcolChecksums As Collection

5  ' the collection of checksums accepted by this model (these don't persist)
Private mcolTempChecksums As Collection

' the Prolog object
Private mudtProlog As Prolog

' needed for I/O
10 Private mblnProcessChecksums As Boolean

' is dirty?
Private mblnIsDirty As Boolean

' needed to save the model one last time after it's frozen
Private mblnFreeze As Boolean

15 Private Enum ModelRecordLayout
    mrLocalDataIndex = 1 ' long (takes 4 bytes)
    mrVariableIndex = 5 ' long
    mrConstraintIndex = 9 ' long
    mrChecksumIndex = 13 ' ' long
    mrLocalData = 51 ' byte
    mrVariables = 201 ' variable length
    ' the constraint data starts wherever the checksum data ends
    ' the checksum data starts wherever the constraint data ends
End Enum

25 Private Sub Class_Initialize()

    Set mudtCVariables = New CVariables
    Set mudtCConstraints = New CConstraints
    Set mudtCClones = New CClones
    Set mcolChecksums = New Collection
    Set mcolTempChecksums = New Collection
    mblnIsDirty = True
    mblnFreeze = False

30

End Sub

Public Property Get FileName() As String

```



```
FileName = mstrDocFN
```

```
End Property
```

```
Public Property Let FileName(ByVal strNewValue As String)
```

```
    mstrDocFN = strNewValue
```

```
5    ' create the FN for the constraint file  
    mstrConFN = left(mstrDocFN, Len(mstrDocFN) - 4) & ".mdl"
```

```
End Property
```

```
Public Property Get IsFrozen() As Boolean
```

```
    IsFrozen = mblnIsFrozen
```

```
10 End Property
```

```
Public Property Let IsFrozen(ByVal blnNewValue As Boolean)
```

```
    mblnIsFrozen = blnNewValue
```

```
End Property
```

```
Public Property Get Comments() As String
```

```
15    Comments = mstrComments
```

```
End Property
```

```
Public Property Let Comments(ByVal strNewValue As String)
```

```
    If mstrComments <> strNewValue Then
```

```
        mstrComments = strNewValue
```

```
20        mblnIsDirty = True
```

```
    End If
```

```
End Property
```

```
Public Property Get Clones() As CClones
```

```
    Set Clones = mudtCClones
```

```
25 End Property
```

```

Public Property Get Variables() As CVariables
    Set Variables = mudtCVariables
End Property

Public Property Get Constraints() As CConstraints
5    Set Constraints = mudtCConstraints
End Property

Public Sub FreezeModel()
    If IsFrozen = False Then
        mblnFreeze = True
10    IsFrozen = True
        WriteModel
    End If
End Sub

Public Sub AddChecksum(ByVal dblChecksum As Double)
15    Call mcolChecksums.Add(dblChecksum)
    mblnIsDirty = True
End Sub

' resets the checksums if this model is a child
Public Sub InitChecksums()
20    Set mcolChecksums = New Collection
End Sub

Private Sub AddTempChecksum(ByVal dblChecksum As Double)
    Call mcolTempChecksums.Add(dblChecksum)
End Sub

25 ' resets the temp checksums if this model is changed and variants are deleted
Public Sub InitTempChecksums()

```

Set mcolTempChecksums = New Collection

End Sub

Public Function ChecksumExists(ByVal dblChecksum As Double) As Boolean

Dim vntChecksum As Variant

5 ' if no variables were checksummed, consider the variant unique

If dblChecksum = 0 Then

ChecksumExists = False

Exit Function

End If

10 ' check the persistent checksums (from accepted or discarded variants)

For Each vntChecksum In mcolChecksums

If vntChecksum = dblChecksum Then

ChecksumExists = True

Exit Function

15 End If

Next vntChecksum

' check the checksums of variants produced in this session

For Each vntChecksum In mcolTempChecksums

If vntChecksum = dblChecksum Then

20 ChecksumExists = True

Exit Function

End If

Next vntChecksum

ChecksumExists = False

25 End Function

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

30 Dim mblnSaved As Boolean

' As frozen models never get saved, they report is dirty

```

' when they are read in from disk. This fix causes them
' to always report not IsDirty.
' If IsFrozen Then
'   IsDirty = False
5 '   Exit Property
' End If

If mdocModel Is Nothing Then
  mblnSaved = True
Else
10   mblnSaved = mdocModel.Saved
End If

If mblnIsDirty Or _
  mudtCVariables.IsDirty Or _
  mudtCConstraints.IsDirty Or _
15   mblnSaved = False Then
  IsDirty = True
Else
  IsDirty = False
End If

20 End Property

Public Property Let LastClone(ByVal intNewValue As Integer)

  mudtCClones.SeqNum = intNewValue

End Property

Public Property Get LastClone() As Integer
25   LastClone = mudtCClones.SeqNum

End Property

' displays model
Public Sub OpenDoc(ByVal udtWord As MSWord)

  Dim udtDS As New DocStatus

30   ' see if word doc is open
  If udtDS.IsOpen(mstrDocFN) = False Then
    Set mdocModel = udtWord.WordApp.Documents.Open(mstrDocFN, , mblnIsFrozen)
  End If

```

```

        mdocModel.Activate

End Sub

' closes model
Public Sub CloseDoc()

5      ' save the model and the word doc
      Call WriteModel

      Dim udtDS As New DocStatus

      ' close the word doc
      If udtDS.IsOpen(mstrDocFN) Then
10          Call mdocModel.Close(False) ' don't save
          Set mdocModel = Nothing
      End If

End Sub

Public Sub CloseAllCloneDocs()

15      Dim udtClone As Clone

      For Each udtClone In mudtCClones
          udtClone.CloseDoc
      Next udtClone

End Sub

20      Public Sub ReadModel()

          Dim udtWAPI As New Win32API

          If udtWAPI.FileExists(mstrConFN) Then
              Set mudtFile = New File
              mudtFile.FileName = mstrConFN
25              mblnProcessChecksums = False
              Call mudtFile.ReadFile(Me, mrLocalDataIndex, mrVariableIndex)
              Call mudtCVariables.ReadCollection(mstrConFN, mrVariableIndex, mrConstraintIndex)
              Call mudtCConstraints.ReadCollection(mstrConFN, mrConstraintIndex,
30              mrChecksumIndex)
              mblnProcessChecksums = True
              Call mudtFile.ReadFile(Me, mrChecksumIndex, READ_UNTIL_EOF)
              Set mudtFile = Nothing

```

```

End If

End Sub

Public Sub ReadObjects()

    Dim vField As Variant

5    If mblnProcessChecksums Then
        On Error GoTo BeatIt
        Do Until err.Number <> 0
            Call mudtFile.ReadField(vField)
            Call mcolChecksums.Add(vField)
10    Loop
    Else
        Call mudtFile.ReadField(vField) ' returns the version stamp
        Call mudtFile.ReadField(vField)
        LastClone = vField
15    Call mudtFile.ReadField(vField)
        IsFrozen = vField
        Call mudtFile.ReadField(vField)
        Comments = vField
    End If

20    BeatIt:

        Exit Sub

End Sub

Public Sub WriteModel()

    Dim lngEndPos As Long
    Dim udtDS As New DocStatus
    Dim udtProg As New Progress

25    If IsDirty = False Then Exit Sub

    ' If IsFrozen And mblnFreeze = False Then Exit Sub

    Call udtProg.Init(2, "Saving the active model...")
30    If udtDS.IsOpen(mstrDocFN) Then ' see if word doc is open
        If Not IsFrozen Then ' command will fail if doc is read-only
            mdocModel.Save
        End If
    End If

```

```

End If
Set mudtFile = New File
mudtFile.FileName = mstrConFN
mblnProcessChecksums = False
5 Call mudtFile.WriteFile(Me, True, mrLocalDataIndex, mrLocalData)
udtProg.Advance
lngEndPos = mudtCVariables.WriteCollection(mstrConFN, mrVariableIndex, mrVariables)
lngEndPos = mudtCConstraints.WriteCollection(mstrConFN, mrConstraintIndex, lngEndPos)
mblnProcessChecksums = True
10 Call mudtFile.WriteFile(Me, False, mrChecksumIndex, lngEndPos)
Set mudtFile = Nothing
udtProg.Advance
IsDirty = False
mblnFreeze = False

15 End Sub

Public Sub WriteObjects()

    Dim vntChecksum As Variant

    If mblnProcessChecksums Then
        For Each vntChecksum In mcolChecksums
            20 Call mudtFile.WriteField(vntChecksum)
        Next vntChecksum
    Else
        Call mudtFile.WriteField(mintVERSIONSTAMP)
        Call mudtFile.WriteField>LastClone)
        25 Call mudtFile.WriteField(IsFrozen)
        Call mudtFile.WriteField(Comments)
    End If

End Sub

' tests the constraints, doesn't care about unique solution
30 Public Function ConstraintsOK(ByVal udtTestType As TestType, _
    ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
    blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

    Dim strVN As String
    Dim strVal As String

    35 Dim udtCS As ConstraintSolver

    Set udtCS = InitConstraintSolver(2, udtTestType)

```

```
udtCS.Prolog = udtProlog
```

```
blnUnderconstrained = False  
blnTestAborted = False
```

```
Select Case udtCS.Solve(srTest)  
  Case srPrologError, srNoSolutions  
    ConstraintsOK = False  
    Exit Function  
  Case srPrologAborted  
    blnTestAborted = True  
    ConstraintsOK = False  
    Exit Function  
  Case srSuccess  
    Do While udtCS.GetNextValue(strUnderconstrainedVN, strVal)  
      If strVal = "_" Then ' it's underconstrained  
        ConstraintsOK = False  
        blnUnderconstrained = True  
        Exit Function  
      End If  
    Loop  
End Select
```

```
ConstraintsOK = True
```

```
End Function
```

```
' implemented in the subclasses of Model
```

```
Public Sub GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _  
  ByVal intNumClones As Integer, ByVal bytDifference As Byte)
```

```
End Sub
```

```
' common code called by GenerateClones in the subclasses
```

```
Public Sub SubstituteValues(ByVal objO As Object, _  
  ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _  
  ByVal intNumClones As Integer, ByVal bytDifference As Byte, _  
  ByVal intStartPos As Integer)
```

```
  Dim udtClone As Clone  
  Dim strPath As String  
  Dim fRange As Range  
  Dim intIndex As Integer  
  Dim udtCS As ConstraintSolver
```



```

Dim udtSortedVs As CVariables
Dim udtCon As Constraint
Dim strVarName As String
Dim strValue As String
5 Dim intTry As Integer
Dim blnSolFound As Boolean
Dim blnUniqueSolFound As Boolean
Dim udtType As VariableType

10 CloseDoc ' close the model doc
CommandBars("File").Controls("Exit").Enabled = False
Randomize

' do substitution of values into model doc

strPath = ExtractPath(FileName)

15 ' Dim udtProgress As New Progress
' Call udtProgress.Init(intNumClones, "Generating variants...")

' initialize the constraint solver
Set udtCS = InitConstraintSolver(bytDifference)
udtCS.Prolog = udtProlog

20 ' solve loop
For intIndex = 1 To intNumClones
    ' try 10x to get a unique sol, then give up
    For intTry = 1 To 10
        DoEvents ' allow abort
        25 If frmProlog.Abort Then
            Exit Sub
        End If
        blnSolFound = False
        blnUniqueSolFound = False
        30 If udtCS.Solve(srGenerate) Then ' found a variant
            blnSolFound = True
        Else
            Exit For
        End If
        35 ' variant found - is it unique?
        If Not ChecksumExists(udtCS.Checksum) Then
            blnUniqueSolFound = True
            Exit For
        End If
        40 Next intTry

```

```

' error if no solution found
If Not blnSolFound Then
    Call MsgBox("No solution could be found for this constraint set", _
        vbExclamation, "Error")
5      '
        udtProgress.Kill
        Exit Sub
End If
' error if unique solution could not be found
If Not blnUniqueSolFound Then
10      Call MsgBox("A unique solution could not be found for this constraint set after 10
attempts." & _
        " You may want to try again.", vbExclamation, "Error")
        udtProgress.Kill
        Exit Sub
15      End If
        ' add the new clone to the collection
        Set udtClone = Clones.Add(ExtractFileName(FileName), True)
        udtClone.Checksum = udtCS.Checksum
        Call AddTempChecksum(udtClone.Checksum)
20      ' add the new clone to the disposition list box
        With frmTCA.lstDisposition
            Call .AddItem(udtClone.FileName)
            .ItemData(.ListCount - 1) = udtClone.index
        End With
25      FileCopy FileName, strPath & udtClone.FileName
        Call udtClone.OpenDoc(udtWord, strPath)
        ' do the substitution
        Set fRange = udtClone.CloneDoc.Content
        fRange.start = intStartPos

30      With fRange.find
        While udtCS.GetNextValue(strVarName, strValue)
            .ClearFormatting
            .Text = strVarName
            .Replacement.ClearFormatting
35            .Replacement.Text = FormatValue(strVarName, strValue)
            ' this first execute needed so Word returns correct value
            .Execute replace:=wdReplaceAll, Forward:=True, _
                MatchCase:=True
        Wend
40      End With

        Dim i, n As Integer
        Dim nShapes As Long

```

```

' n = udtClone.CloneDoc.InlineShapes.Count
'
' For i = 1 To n
'     udtCS.ResetValueIndex
5 '
'     While udtCS.GetNextValue(strVarName, strValue)
'         udtClone.CloneDoc.InlineShapes(i).Select
'         Call MTTTextSubstitution(strVarName, strValue)
'     Wend
10 ' Next

udtClone.CloneDoc.Bookmarks("stem1").Range.Copy

If udtClone.CloneDoc.Bookmarks.Exists("tca_Stem") = True Then
    Dim stemRange As Range
    Set stemRange = udtClone.CloneDoc.Bookmarks("tca_Stem").Range
15 stemRange.Paste
    udtClone.CloneDoc.Bookmarks.Add name:="tca_Stem", Range:=stemRange
Else
    Call MsgBox("Model is missing TCA_Stem Bookmark!", vbExclamation, "Hey!")
End If

20 ' trim hard returns at end of stem
Dim retchr As String
retchr = Chr$(13)

With stemRange
    n = 0
    i = .Words.Count
25
    While .Words(i).Text = retchr And i > 1 ' Rob: I added the And part. Pete
        i = i - 1
        If .Words(i).Text = retchr Then
            n = n + 1
30        End If
    Wend

    If n > 0 Then
        .Words(.Words.Count - n + 1).Delete Count:=n
    End If
35 End With

' callback to subclass to code unique to this model type
Call objO.CreateVariant(udtClone)
udtProgress.Advance

```

```
        udtClone.CloseDoc
    Next intIndex
```

```
End Sub
```

```
' create, initialize constraint solver
```

```
5 Private Function InitConstraintSolver(ByVal bytDifference As Byte, _
    Optional ByVal udtTestType As TestType = tcTestAll) As ConstraintSolver
```

```
    Dim udtVar As Variable
    Dim udtCon As Constraint
    Dim udtVarString As VarString
10 Dim udtCS As New ConstraintSolver
    Dim udtSortedVs As CVariables
```

```
    ' add enabled variables to ConstraintSolver object, sorted by length,
    ' strings first
```

```
15 Set udtSortedVs = muddCVariables.SortVarNamesByLength
```

```
    For Each udtVar In udtSortedVs
        If udtVar.Enabled Then
            Call udtCS.AddVariable(udtVar)
        End If
20 Next udtVar
```

```
    ' Add enabled constraints
    For Each udtCon In Constraints
        If udtCon.Enabled Then
            If udtTestType = tcTestAll Or _
25 udtCon.ConstraintType = udtTestType - 1 Then
                Call udtCS.AddConstraint(udtCon)
            End If
        End If
    Next udtCon
```

```
30 udtCS.DiffWeight = bytDifference
```

```
    Set InitConstraintSolver = udtCS
```

```
End Function
```

```
' formats all math variables for item presentation
```

```
35 Private Function FormatValue(ByVal strVarName As String, _
    ByVal strValue As String) As String
```

```
Dim udtV As Variable
Dim udtVR As VarReal
Dim udtVF As VarFraction
```

```
For Each udtV In mudtCVariables
```

```
    If udtV.Enabled Then
```

```
        If udtV.name = ExtractVarName(strVarName) Then
```

```
            Select Case udtV.Type
```

```
                Case vtInteger
```

```
                    FormatValue = strValue
```

```
                Case vtReal
```

```
                    Set udtVR = udtV
```

```
                    FormatValue = FormatReal(strValue, _
                        udtVR.DecimalPlaces, udtVR.TrailingZeros)
```

```
                Case vtFraction
```

```
                    Set udtVF = udtV
```

```
                    If udtVF.MixedNumbers Then
```

```
                        FormatValue = FormatFraction(strValue)
```

```
                    Else
```

```
                        FormatValue = strValue
```

```
                    End If
```

```
                Case vtString
```

```
                    FormatValue = strValue
```

```
                Case vtUntyped
```

```
                    FormatValue = FormatUntyped(strValue)
```

```
            End Select
```

```
        Exit For
```

```
    End If
```

```
End If
```

```
Next udtV
```

```
End Function
```

' takes the index off of a string variable name that is indexed

```
Private Function ExtractVarName(ByVal strName As String) As String
```

```
    Dim varI As Variant
```

```
    varI = InStr(1, strName, ".")
```

```
    If varI > 0 Then
```

```
        ExtractVarName = left(strName, varI - 1)
```

```
    Else
```

```
        ExtractVarName = strName
```

```
    End If
```

End Function

' formats reals for item presentation

Private Function FormatReal(ByVal strReal As String, ByVal intPlaces As Integer, _
ByVal blnTZeros As Boolean) As String

```
5    Dim varPos As Variant
    Dim intLen As Integer
    Dim strI As String
    Dim strD As String
    Dim blnZeroFound As Boolean

10   varPos = InStr(1, strReal, ".")

    ' isolate strings on either side of decimal point
    If varPos = 0 Then
        strI = strReal
    Else
15     strI = Mid(strReal, 1, varPos - 1)
        strD = Mid(strReal, varPos + 1, Len(strReal))
    End If

    intLen = Len(strD)

    ' pad or trim to intPlaces
20   If intLen < intPlaces Then
        strD = strD & String(intPlaces - intLen, "0")
    Else
        If intLen > intPlaces Then
            strD = left(strD, intPlaces)
25         End If
    End If

    ' get rid of trailing zeros if desired
    If blnTZeros = False Then
        Do
30         blnZeroFound = False
            If right(strD, 1) = "0" Then
                strD = left(strD, Len(strD) - 1)
                blnZeroFound = True
            End If
35         Loop While blnZeroFound
    End If

    ' reassemble string
```

```

If Len(strD) > 0 Then
    FormatReal = strI & "." & strD
Else
    FormatReal = strI
End If

```

End Function

' formats fraction as mixed number for item presentation
Private Function FormatFraction(ByVal strFraction As String) As String

```

Dim intNum As Integer
Dim intDen As Integer
Dim intQuot As Integer
Dim vntI As Variant

```

```

vntI = InStr(strFraction, "/")

```

```

' it's an integer
If vntI = 0 Then ' it's a whole number
    FormatFraction = strFraction
    Exit Function
End If

```

```

intNum = CInt(left(strFraction, vntI - 1))
intDen = CInt(right(strFraction, Len(strFraction) - vntI))

```

```

If intDen > 0 And Abs(intNum) > intDen Then
    intQuot = Int(intNum / intDen)
    intNum = intNum Mod intDen
    FormatFraction = Trim(Str(intQuot)) & " " & Trim(Str(Abs(intNum))) & "/" & _
        Trim(Str(intDen))

```

```

Else
    FormatFraction = strFraction
End If

```

End Function

Private Function FormatUntyped(ByVal strValue As String)

```

Dim varI As Variant

```

```

' see if the value is a list - if so, it will be in []
If left(strValue, 1) = "[" And right(strValue, 1) = "]" Then
    ' trim the brackets off

```

```

    FormatUntyped = Mid(strValue, 2, Len(strValue) - 2)
Else
    FormatUntyped = strValue
End If

```

```

5 End Function

```

```

Private Function MTTextSubstitution(Source As String, dest As String)
    Dim stat

```

```

    Selection.Copy

```

```

    'Init API, reset transform
10 If MTUtil.CheckMTDLLVersion = 0 Then Exit Function
    MTXFormReset

```

```

    'first substitution
    stat = MTXFormAddVarSub( _
15     mtxfmSUBST_ALL, _
        mtxfmVAR_SUB_PLAIN_TEXT, Source, 0, _
        mtxfmVAR_SUB_PLAIN_TEXT, dest, Len(dest), mtxfmSTYLE_NUMBER)

```

```

    If stat <> 0 Then
        MsgBox "MTXFormAddVarSub returned: " + Str(stat)
        Exit Function
20    End If

```

```

    'do the substitution
    stat = TransformGraphicEquation
    If stat <> 0 Then
        MsgBox "TransformGraphicEquation returned: " + Str(stat)
25        Exit Function
    End If

```

```

    MTTermAPI

```

```

    Selection.Delete

```

```

    'Paste new equation
30    Selection.Collapse Direction:=wdCollapseEnd
    Selection.PasteSpecial Placement:=wdInLine

```

```

End Function

```



```

' PrintModel.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "PrintModel"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private mstrModelName As String
    Private mstrNow As String
    Private mintPage As Integer
15  Private mintTab As Integer

    Public Property Let ModelName(ByVal strNewValue As String)

        mstrModelName = strNewValue

    End Property

20  Public Sub PrintString(ByVal strS As String, ByVal intIndent As Integer)

        CheckPageBreak

        If Printer.CurrentY = 0 Then PrintHeading

25  Printer.Print Space(intIndent * mintTab) & strS

    End Sub

    Private Sub PrintHeading()

        Dim intY As Integer

30  Printer.CurrentY = 1440 ' top margin
        Printer.Print Space(mintTab) & _
            "Variables and constraints for model " & mstrModelName
        Printer.Print Space(mintTab) & mstrNow
        Printer.CurrentY = Printer.CurrentY + 100
35  Printer.Line Step(0, 0)-Step(Printer.Width, 0)
        SkipLine

```

```

intY = Printer.CurrentY
Printer.CurrentY = Printer.Height - 1700
Printer.Line Step(0, 0)-Step(Printer.Width, 0)
Printer.CurrentY = Printer.CurrentY + 100
5 Printer.CurrentX = 0
Printer.Print Space(mintTab) & "Page " & Str(mintPage)
Printer.CurrentY = intY
mintPage = mintPage + 1

```

End Sub

```

10 Private Sub SkipLine()

```

```

    Printer.Print " "

```

End Sub

```

Private Sub CheckPageBreak()

```

```

15 Select Case Printer.PaperSize
    Case vbPRPSLetter, vbPRPSLetterSmall
        Call CheckOrientation(8.5, 11)
    Case vbPRPSTabloid
        Call CheckOrientation(11, 17)
20 Case vbPRPSLedger
        Call CheckOrientation(17, 11)
    Case vbPRPSLegal
        Call CheckOrientation(8.5, 14)
End Select

```

```

25 End Sub

```

```

Private Sub CheckOrientation(ByVal sngWidth As Single, _
    ByVal sngHeight As Single)

```

```

' convert inches to twips
30 sngWidth = sngWidth * 1440
    sngHeight = sngHeight * 1440

```

```

If Printer.Orientation = vbPRORPortrait Then
    If Printer.CurrentY >= sngHeight - 2200 Then
        Printer.NewPage
35 End If
Else
    If Printer.CurrentY >= sngWidth - 2200 Then

```

```
Printer.NewPage
End If
End If
```

```
5 End Sub
```

```
Private Sub Class_Initialize()
```

```
Printer.FontSize = 11
mstrNow = Now
mintPage = 1
10 mintTab = 4
```

```
End Sub
```

```
Private Sub Class_Terminate()
```

```
Printer.EndDoc
15 End Sub
```

```

' Progress.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "Progress"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
' class to give visual indication of progress
Option Explicit

Private mintStepSize As Integer

' pulls up form
15 Public Sub Init(ByVal intNumIncrements As Integer, _
    Optional ByVal strCaption As String)

    If intNumIncrements = 0 Then ' prevent divide by 0
        Beep
20     Exit Sub
    End If

    mintStepSize = 500 / intNumIncrements
    frmProgress.prbProgressBar.Max = mintStepSize * intNumIncrements

25     If Len(strCaption) > 0 Then
        frmProgress.lblProgress = strCaption
    End If

30     frmProgress.Show
    frmProgress.Refresh

End Sub

' bumps the progress bar to the next increment. When the progress
35 ' bar is fully advanced, the form is unloaded.
Public Sub Advance()

    Dim intStop As Integer

    With frmProgress.prbProgressBar
40         If .Value = .Max Then

```

```

        Exit Sub
    End If
    intStop = .Value + mintStepSize
    Do Until .Value = intStop
5       .Value = .Value + 1
        If .Value = .Max Then
            Unload frmProgress
            Exit Sub
        End If
10      Loop
    End With

End Sub

Public Sub AbsoluteAdvance(ByVal intNewValue As Integer)

15      frmProgress.prbProgressBar.Value = intNewValue * mintStepSize

End Sub

Public Sub Kill()

    Unload frmProgress
20 End Sub

```

```
' Prolog.cls
VERSION 1.0 CLASS
BEGIN
```

```
MultiUse = 0 'False
Persistable = 0 'NotPersistable
DataBindingBehavior = 0 'vbNone
DataSourceBehavior = 0 'vbNone
MTSTransactionMode = 0 'NotAnMTSObject
END
```

```
Attribute VB_Name = "Prolog"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit
```

```
Private Declare Function StartProlog4Session Lib "prlghlapi.dll" _
    (ByVal strP4FN As String) As Long
Private Declare Function EndProlog4Session Lib "prlghlapi.dll" () As Long
Private Declare Function GetHLAPIVersion Lib "prlghlapi.dll" () As String
Private Declare Function VBGetHLAPIVersion Lib "prlghlapi.dll" () As String
Private Declare Function SolveConstraintOrdered Lib "prlghlapi.dll" _
    (ByVal Constraint As String, ByVal SolutionOrder As Long) As Long
Private Declare Function SolveConstraintRandomly Lib "prlghlapi.dll" _
    (ByVal Constraint As String) As Long
Private Declare Function SolveConstraintOrderedNSolns Lib "prlghlapi.dll" _
    (ByVal Constraint As String, ByVal SolutionOrder As Long, _
    ByVal NumSols As Long) As Long
Private Declare Function IsFullyConstrained Lib "prlghlapi.dll" _
    (ByVal Constraint As String) As Long
Private Declare Function GetValue Lib "prlghlapi.dll" _
    (ByVal strVarName As String) As Long
Private Declare Function VBGetValue_string Lib "prlghlapi.dll" _
    (ByVal udtPtr As Any) As String
Private Declare Function VBPrintAllVarVals Lib "prlghlapi.dll" () As String
Private Declare Function SetSolnDiffWt Lib "prlghlapi.dll" _
    (ByVal Weight As Long) As Long
Private Declare Function SetPrologInterruptFile Lib "prlghlapi.dll" _
    (ByVal strFN As String) As Long
```

```
'Keep the constants in sync with appropriate values in prlghlapi.h
' Solution-Orders:
```

Private Enum PrologOrder

prDontCareOrder = 0

prDifferentOrder = 10

prLikeOrder = 20

prRandomOrder = 30

prUniqueOrder = 40

End Enum

Private Enum PrologType

prValUnknown = 0

prValInteger = 10

prValRationalFloat = 12

prValRationalFraction = 13

prValIrrational = 14

prValReal = 15

prValString = 20

prValList = 25

prValFunctor = 30

prValSymbol = 35

prValVar = 100

End Enum

Private Enum PrologErrors

prErrInitialization = -10

prErrIntegerraintTooLong = -15

prErrGettingTerm = -20

prErrMakingFunctor = -25

prErrInvalidInterval = -30

prErrArityTooMany = -35

prErrParse = -40

prErrNullTerm = -45

End Enum

' used to hold all strings for the Prolog

Private mcolVNs As Collection

Private mstrDelimit As String

Private mintNumSols As Integer

Event Finished(ByVal lngRet As Long)

Private Sub Class_Initialize()

Set mcolVNs = New Collection


```

    Set gProlog = Me ' gProlog is defined in Timer.bas

    Dim lngRet As Long
5
    ' if this file exists, interrupt prolog processing
    lngRet = SetPrologInterruptFile("c:\halt.tca")

End Sub

10 Private Sub Class_Terminate()

    Set gProlog = Nothing

End Sub

Public Property Get Version() As String

    Version = GetHLAPIVersion()
15
End Property

' sets the degree of difference in the variants. Range is 0 to 2.
Public Property Let DiffWeight(ByVal bytDifference As Byte)

20    Call SetSolnDiffWt(CLng(bytDifference))

End Property

Public Function StartProlog() As Boolean

    ChDir App.Path ' set path to application dir for hlp4lib.p4 file
25    StartProlog = CBool(StartProlog4Session("hlp4lib.p4"))

End Function

Public Function EndProlog() As Boolean

    ChDir App.Path ' set path to application dir for hlp4lib.p4 file
30    EndProlog = CBool(EndProlog4Session())

End Function

Public Sub AddVariable(ByVal strS As String)

```

```
    If Len(strS) > 0 Then ' it's not an untyped variable
        Call mcolVNs.Add(strS)
        mstrDelimit = "end_var_defs,"
    End If
```

```
5    End Sub
```

```
Public Sub AddConstraint(ByVal strS As String)
```

```
    Call mcolVNs.Add(mstrDelimit & strS)
    mstrDelimit = ""
```

```
End Sub
```

```
10 Public Sub SolveConstraintsRandomly()
```

```
    SolveAsync ' in Timer.bas - must be in a standard module
```

```
End Sub
```

```
Public Sub SolveConstraintsAsync()
```

```
15    Dim strS As String
    Dim lngRet As Long
```

```
    lngRet = -1 ' default to error condition
```

```
20    If mcolVNs.Count > 0 Then ' there's something for Prolog to chew on
        strS = BuildString()
        ChDir App.Path ' set path to application dir for hlp4lib.p4 file
        lngRet = SolveConstraintRandomly(strS) ' call Prolog
25    End If
```

```
    RaiseEvent Finished(lngRet)
```

```
    Set mcolVNs = New Collection
```

```
30 End Sub
```

```
Private Function RandomNumSols() As Integer
```

```
    Randomize
    RandomNumSols = 10 * Rnd - 0.5
35    If RandomNumSols = 0 Then RandomNumSols = 1
```

End Function

Private Sub Advance(ByVal lngRet As Long)

Dim intI As Integer

5 For intI = 1 To lngRet
NextSolution
Next intI

End Sub

10 ' gets the next solution, returns true if one exists, false if it doesn't
Private Function NextSolution() As Boolean

ChDir App.Path ' set path to application dir for hlp4lib.p4 file
NextSolution = SolveConstraintOrderedNSolns(vbNullString, _
prUniqueOrder, mintNumSols)

15 End Function

Public Property Get PrintAllVals() As String

PrintAllVals = VBPrintAllVarVals

20 End Property

' get the values associated with each solution

Public Property Get Value(ByVal strVN As String) As String

25 Dim lngPtr As Long
Dim strT As String

ChDir App.Path ' set path to application dir for hlp4lib.p4 file
lngPtr = GetValue(strVN) ' returns a pointer to the variable

30 If lngPtr Then ' to handle untyped variables that have no constraint, and therefore no value
strT = VBGetValue_string(lngPtr) ' returns a string
Value = Left(strT, Len(strT) - 1) ' trim off the null delimiter

Else

Value = " _"

35 End If

End Property

Private Function BuildString() As String

Dim varStr As Variant

Dim strS As String

For Each varStr In mcolVNs

5 strS = strS & varStr & ", "

Next varStr

' trim off the last comma and space

strS = Left(strS, Len(strS) - 2)

10 ' add a period

strS = strS & "."

BuildString = strS

15 End Function

Public Sub ShowString()

Dim strS As String

strS = BuildString()

20 ' Call MsgBox(strS, , "Prolog string is:")

End Sub

```

' PSMODEL.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "SMCModel"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    Implements Model

    Dim mudtModel As Model
    Dim lastStart As Integer

    Private Sub Class_Initialize()
20        Set mudtModel = New Model
    End Sub

    ' Delegated to Class Model
    Public Property Get Model_FileName() As String

        Model_FileName = mudtModel.FileName
25    End Property

    ' Delegated to Class Model
    Public Property Let Model_FileName(ByVal strNewValue As String)

        mudtModel.FileName = strNewValue

    End Property

30    ' Delegated to Class Model
    Public Property Get Model_IsFrozen() As Boolean

```

```
Model_IsFrozen = mudtModel.IsFrozen
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)
```

```
5      mudtModel.IsFrozen = blnNewValue
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)
```

```
    Call mudtModel.AddChecksum(dblChecksum)
```

```
10 End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_InitChecksums()
```

```
    mudtModel.InitChecksums
```

```
End Sub
```

```
15 ' Delegated to Class Model
```

```
Public Sub Model_InitTempChecksums()
```

```
    mudtModel.InitTempChecksums
```

```
End Sub
```

```
' Delegated to Class Model
```

```
20 Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean
```

```
    Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)
```

```
End Function
```

```
' Delegated to Class Model
```

```
Public Property Get Model_Comments() As String
```

```
25      Model_Comments = mudtModel.Comments
```

```
End Property
```

```
' Delegated to Class Model
Public Property Let Model_Comments(ByVal strNewValue As String)
```

```
    mudtModel.Comments = strNewValue
```

```
End Property
```

```
5 ' Delegated to Class Model
Public Property Get Model_Clones() As CClones
```

```
    Set Model_Clones = mudtModel.Clones
```

```
End Property
```

```
10 ' Delegated to Class Model
Public Property Get Model_Variables() As CVariables
```

```
    Set Model_Variables = mudtModel.Variables
```

```
End Property
```

```
' Delegated to Class Model
Public Property Get Model_Constraints() As CConstraints
```

```
15 Set Model_Constraints = mudtModel.Constraints
```

```
End Property
```

```
' Delegated to Class Model
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)
```

```
    mudtModel.IsDirty = blnNewValue
```

```
20 End Property
```

```
' Delegated to Class Model
Public Property Get Model_IsDirty() As Boolean
```

```
    Model_IsDirty = mudtModel.IsDirty
```

```
End Property
```

```
25 ' Delegated to Class Model
Public Property Let Model_LastClone(ByVal intNewValue As Integer)
```

```
    mudtModel.LastClone = intNewValue
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Property Get Model_LastClone() As Integer
```

```
5    Model_LastClone = mudtModel.LastClone
```

```
End Property
```

```
' Delegated to Class Model
```

```
Public Sub Model_FreezeModel()
```

```
    Call mudtModel.FreezeModel
```

```
10 End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_OpenDoc(ByVal udtWord As MSWord)
```

```
    Call mudtModel.OpenDoc(udtWord)
```

```
End Sub
```

```
15 ' Delegated to Class Model
```

```
Public Sub Model_CloseDoc()
```

```
    Call mudtModel.CloseDoc
```

```
End Sub
```

```
' Delegated to Class Model
```

```
20 Public Sub Model_CloseAllCloneDocs()
```

```
    Call mudtModel.CloseAllCloneDocs
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_ReadModel()
```

```
25    mudtModel.ReadModel
```

```
End Sub
```



```
' Delegated to Class Model
Public Sub Model_ReadObjects()
```

```
    mudtModel.ReadObjects
```

```
End Sub
```

```
5 ' Delegated to Class Model
Public Sub Model_WriteModel()
```

```
    mudtModel.WriteModel
```

```
End Sub
```

```
10 ' Delegated to Class Model
Public Sub Model_WriteObjects()
```

```
    mudtModel.WriteObjects
```

```
End Sub
```

```
15 ' Delegated to Class Model
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
    ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
    blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean

    Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _
        blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)
```

```
End Function
```

```
20 ' implemented here
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
    ByVal intNumClones As Integer, ByVal bytDifference As Byte)
```

```
    Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _
        bytDifference, 50)
```

```
25 End Sub
```

```
' Delegated to Class Model
Public Sub Model_SubstituteValues(ByVal objO As Object, _
    ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
    ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
30 ByVal intStartPos As Integer)
```

End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

With udtClone.CloneDoc.Bookmarks

If .Exists("tca_RespA") = False Or _
.Exists("tca_RespB") = False Or _
.Exists("tca_RespC") = False Or _
.Exists("tca_RespD") = False Or _
.Exists("tca_RespE") = False Or _
.Exists("tca_Key") = False Then

Call MsgBox("Model is missing a TCA Bookmark!", vbExclamation, "Hey!")

Exit Sub

End If

End With

Dim nchoices As Integer

Dim lowerbound As Integer

Dim upperbound As Integer

nchoices = 5

lowerbound = 1

upperbound = 8

Dim resp(10) As String

Dim used(10) As Integer

resp(0) = udtClone.CloneDoc.Bookmarks("key").Range.Text

Dim i As Integer

For i = lowerbound To upperbound

used(i) = 0

resp(i) = udtClone.CloneDoc.Bookmarks("resp" & Format(i)).Range.Text

Next

Dim nselected As Integer

nselected = 0

Dim rnumber As Integer

Dim rnumbers(10) As Integer

While (nselected < upperbound)

rnumber = (upperbound - lowerbound + 1) * Rnd + lowerbound - 0.5

If (rnumber > upperbound) Then

```
    number = upperbound  
End If
```

```
    If (used(rnumber) = 0) Then  
        used(rnumber) = 1  
        nselected = nselected + 1  
        rnumbers(nselected) = number  
    End If  
Wend
```

```
Dim unsorted(10) As Integer  
unsorted(0) = 0  
nselected = 0
```

```
Dim j As Integer  
Dim n As Integer
```

```
Dim crStr As String  
Dim tabcrStr As String  
crStr = Chr(13)  
tabcrStr = Chr(9) & Chr(13)
```

```
For i = lowerbound To upperbound  
    If resp(rnumbers(i)) <> tabcrStr And _  
        resp(rnumbers(i)) <> crStr And _  
        Mid(resp(rnumbers(i)), 1, 10) <> "Distractor" Then
```

```
        n = 0
```

```
        For j = 0 To nselected
```

```
            If IsNumeric(resp(rnumbers(i))) = True And _
```

```
                IsNumeric(resp(unsorted(j))) = True And _
```

```
                Asc(resp(rnumbers(i))) <> 36 Then ' 36 is the $ sign
```

```
                If Val(resp(rnumbers(i))) = Val(resp(unsorted(j))) Then
```

```
                    If Asc(resp(rnumbers(i))) <> 1 Then
```

```
                        n = 1
```

```
                    Exit For
```

```
                End If
```

```
            End If
```

```
        Else
```

```
            If resp(rnumbers(i)) = resp(unsorted(j)) Then
```

```
                If Asc(resp(rnumbers(i))) <> 1 Then
```

```
                    n = 1
```

```
                Exit For
```

```
            End If
```

```
        End If
```

End If
Next

If n = 0 Then
nselected = nselected + 1
unsorted(nselected) = rnumbers(i)
If nselected = nchoices - 1 Then
If nselected = upperbound Then
Exit For
End If
End If
End If

Next

For i = 0 To nselected
used(i) = 0
Next

Dim sorted(10) As Integer
Dim resp1, resp2 As String
Dim val1, val2 As Variant

For i = 0 To nselected
For j = 0 To nselected
If (used(j) = 0) Then
sorted(i) = unsorted(j)
n = j
Exit For
End If
Next

For j = 0 To nselected
If (used(j) = 0) Then

resp1 = resp(unsorted(j))
resp2 = resp(sorted(i))

If left(resp1, 1) = "\$" Then
val1 = Val(right(resp1, Len(resp1) - 1))
Else
val1 = Val(resp1)
End If

If left(resp2, 1) = "\$" Then
val2 = Val(right(resp2, Len(resp2) - 1))

```
Else
    val2 = Val(resp2)
End If
```

```
5      If (val1 < val2) Then
        sorted(i) = unsorted(j)
        n = j
    End If
```

```
    End If
Next
```

```
10     used(n) = 1
Next
```

```
For i = 0 To nselected
    If sorted(i) = 0 Then
        Exit For
    End If
Next
```

```
Dim min, max As Integer
```

```
min = i - 4
If min < 0 Then
    min = 0
End If
```

```
max = i
If max > nselected - 4 Then
    max = nselected - 4
End If
```

```
If max < 0 Then
    max = 0
End If
```

```
30     Dim iStart As Integer
        Dim iEnd As Integer
```

```
If max > 0 And max + 4 <= nselected Then
    iStart = lastStart
    While iStart = lastStart
        iStart = (max - min + 1) * Rnd + min - 0.5
    Wend
```

```
    lastStart = iStart  
    iEnd = iStart + nchoices - 1
```

```
Else
```

```
    iStart = 0
```

```
    If nselected > 4 Then
```

```
        iEnd = 4
```

```
    Else
```

```
        iEnd = nselected
```

```
    End If
```

```
    lastStart = iStart
```

```
End If
```

```
Dim respRange As Range
```

```
Dim choice As String
```

```
Dim key As String
```

```
n = 1
```

```
For i = iStart To iEnd
```

```
    choice = Mid("ABCDE", n, 1)
```

```
    If sorted(i) = 0 Then
```

```
        udtClone.CloneDoc.Bookmarks("key").Range.Copy
```

```
    Else
```

```
        udtClone.CloneDoc.Bookmarks("resp" & Format(sorted(i))).Range.Copy
```

```
    End If
```

```
    Set respRange = udtClone.CloneDoc.Bookmarks("tca_Resp" & choice).Range
```

```
    respRange.Paste
```

```
    respRange.Borders.Enable = False
```

```
    respRange.Borders.InsideLineStyle = wdLineStyleNone
```

```
    udtClone.CloneDoc.Bookmarks.Add name:="tca_Resp" & choice, Range:=respRange
```

```
    respRange.InsertBefore Text:=choice & ". "
```

```
    If sorted(i) = 0 Then
```

```
        key = choice
```

```
        udtClone.key = choice
```

```
    End If
```

```
    n = n + 1
```

```
Next
```

```
For i = nselected + 1 To nchoices - 1
```



```

' QCModel.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "QCModel"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = False
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    Implements Model

    Dim mudtModel As Model

    Private Sub Class_Initialize()

        Set mudtModel = New Model

20    End Sub

    ' Delegated to Class Model
    Public Property Get Model_FileName() As String

        Model_FileName = mudtModel.FileName

    End Property

25    ' Delegated to Class Model
    Public Property Let Model_FileName(ByVal strNewValue As String)

        mudtModel.FileName = strNewValue

    End Property

    ' Delegated to Class Model
30    Public Property Get Model_IsFrozen() As Boolean

        Model_IsFrozen = mudtModel.IsFrozen

```


End Property

' Delegated to Class Model

Public Property Let Model_IsFrozen(ByVal blnNewValue As Boolean)

 mudtModel.IsFrozen = blnNewValue

5 End Property

' Delegated to Class Model

Public Property Get Model_Comments() As String

 Model_Comments = mudtModel.Comments

End Property

10 ' Delegated to Class Model

Public Property Let Model_Comments(ByVal strNewValue As String)

 mudtModel.Comments = strNewValue

End Property

15 ' Delegated to Class Model

Public Property Get Model_Clones() As CClones

 Set Model_Clones = mudtModel.Clones

End Property

' Delegated to Class Model

Public Property Get Model_Variables() As CVariables

20 Set Model_Variables = mudtModel.Variables

End Property

' Delegated to Class Model

Public Property Get Model_Constraints() As CConstraints

 Set Model_Constraints = mudtModel.Constraints

25 End Property

'Delegated to Class Model

```
Public Sub Model_AddChecksum(ByVal dblChecksum As Double)
```

```
    Call mudtModel.AddChecksum(dblChecksum)
```

```
End Sub
```

```
' Delegated to Class Model
```

```
5 Public Sub Model_InitChecksums()
```

```
    mudtModel.InitChecksums
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_InitTempChecksums()
```

```
10    mudtModel.InitTempChecksums
```

```
End Sub
```

```
'Delegated to Class Model
```

```
Public Function Model_ChecksumExists(ByVal dblChecksum As Double) As Boolean
```

```
    Model_ChecksumExists = mudtModel.ChecksumExists(dblChecksum)
```

```
15 End Function
```

```
' Delegated to Class Model
```

```
Public Property Let Model_IsDirty(ByVal blnNewValue As Boolean)
```

```
    mudtModel.IsDirty = blnNewValue
```

```
End Property
```

```
20 ' Delegated to Class Model
```

```
Public Property Get Model_IsDirty() As Boolean
```

```
    Model_IsDirty = mudtModel.IsDirty
```

```
End Property
```

```
' Delegated to Class Model
```

```
25 Public Property Let Model_LastClone(ByVal intNewValue As Integer)
```

```
    mudtModel.LastClone = intNewValue
```

End Property

' Delegated to Class Model

Public Sub Model_FreezeModel()

Call mudtModel.FreezeModel

5 End Sub

' Delegated to Class Model

Public Property Get Model_LastClone() As Integer

Model_LastClone = mudtModel.LastClone

End Property

10

' Delegated to Class Model

Public Sub Model_OpenDoc(ByVal udtWord As MSWord)

Call mudtModel.OpenDoc(udtWord)

End Sub

' Delegated to Class Model

15 Public Sub Model_CloseDoc()

Call mudtModel.CloseDoc

End Sub

' Delegated to Class Model

Public Sub Model_CloseAllCloneDocs()

20 Call mudtModel.CloseAllCloneDocs

End Sub

' Delegated to Class Model

Public Sub Model_ReadModel()

mudtModel.ReadModel

25 End Sub

' Delegated to Class Model

```
Public Sub Model_ReadObjects()
```

```
    mudtModel.ReadObjects
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_WriteModel()
```

```
    mudtModel.WriteModel
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_WriteObjects()
```

```
    mudtModel.WriteObjects
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Function Model_ConstraintsOK(ByVal udtTestType As TestType, _
```

```
    ByVal udtProlog As Prolog, blnUnderconstrained As Boolean, _
```

```
    blnTestAborted As Boolean, strUnderconstrainedVN As String) As Boolean
```

```
    Model_ConstraintsOK = mudtModel.ConstraintsOK(udtTestType, udtProlog, _  
        blnUnderconstrained, blnTestAborted, strUnderconstrainedVN)
```

```
End Function
```

```
' implemented here
```

```
Public Sub Model_GenerateClones(ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _  
    ByVal intNumClones As Integer, ByVal bytDifference As Byte)
```

```
    Call mudtModel.SubstituteValues(Me, udtWord, udtProlog, intNumClones, _  
        bytDifference, 275)
```

```
End Sub
```

```
' Delegated to Class Model
```

```
Public Sub Model_SubstituteValues(ByVal objO As Object, _
```

```
    ByVal udtWord As MSWord, ByVal udtProlog As Prolog, _
```

```
    ByVal intNumClones As Integer, ByVal bytDifference As Byte, _
```

```
    ByVal intStartPos As Integer)
```

End Sub

Public Sub CreateVariant(ByVal udtClone As Clone)

Dim rnumber As Integer

Dim sLen As Integer

5 Dim columnRange As Range

Dim columnAValStr As String

Dim columnBValStr As String

With udtClone.CloneDoc

10 rnumber = .Tables(2).Rows.Count * Rnd + 0.5

.Tables(2).Cell(Row:=rnumber, Column:=1).Range.Copy

columnAValStr = .Tables(2).Cell(Row:=rnumber, Column:=2).Range.Text

sLen = Len(columnAValStr)

If sLen > 1 Then

15 columnAValStr = left(columnAValStr, sLen - 1)

End If

Set columnRange = .Bookmarks("tca_ColumnA").Range

columnRange.Paste

rnumber = .Tables(3).Rows.Count * Rnd + 0.5

20 .Tables(3).Cell(Row:=rnumber, Column:=1).Range.Copy

columnBValStr = .Tables(3).Cell(Row:=rnumber, Column:=2).Range.Text

sLen = Len(columnBValStr)

If sLen > 1 Then

25 columnBValStr = left(columnBValStr, sLen - 1)

End If

Set columnRange = .Bookmarks("tca_ColumnB").Range

columnRange.Paste

If .Tables(1).Columns.Count = 4 Then ' fixes weird behavior if only 1 row in model

.Tables(1).Cell(Row:=1, Column:=4).Delete

30 .Tables(1).Cell(Row:=1, Column:=3).Delete

End If

Dim key As String

Dim columnAValue

Dim columnBValue

```
If IsNumeric(columnAValStr) = True And _  
    IsNumeric(columnBValStr) = True Then
```

```
    columnAValue = Val(columnAValStr)  
    columnBValue = Val(columnBValStr)
```

```
5      If columnAValue > columnBValue Then  
        key = "A"  
      ElseIf columnBValue > columnAValue Then  
        key = "B"  
      ElseIf columnAValue = columnBValue Then  
10     key = "C"  
      End If
```

```
End If
```

```
End With
```

```
Dim keyRange As Range
```

```
15 Set keyRange = udtClone.CloneDoc.Bookmarks("tca_Key").Range
```

```
If key = "" Then
```

```
    keyRange.InsertBefore Text:="TCA cannot determine the key"
```

```
Else
```

```
20     keyRange.InsertBefore Text:="Key is " & key
```

```
End If
```

```
    udtClone.key = key
```

```
End Sub
```

```

' StringSolver.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = 0 'False
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "StringSolver"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Option Explicit

    Dim mudtVS As VarString

    Dim mcolValues As Collection

    Public Property Let StringVariable(ByVal udtNewValue As VarString)

        Set mudtVS = udtNewValue
20    End Property

    Public Property Get RandomValueCollection() As Collection

        Dim udtSS As SubString
        Dim strS As String
25        Dim varS As Variant

        Set mcolValues = New Collection

        strS = mudtVS.StringCollection.Item(GetRandomIndex)

30        If mudtVS.IsIndexed Then
            Set udtSS = New SubString
            udtSS.Delimiter = mudtVS.Delimiter
            udtSS.StringValue = strS
35            For Each varS In udtSS.StringCollection
                Call mcolValues.Add(varS)
            Next varS
        Else

```

```
    Call mcolValues.Add(strS)
End If
```

```
    Set RandomValueCollection = mcolValues
```

```
5 End Property
```

```
Private Function GetRandomIndex() As Integer
```

```
    Dim intI As Integer
```

```
    intI = mudtVS.StringCollection.Count * Rnd + 0.5
```

```
10 ' Seems to produce an out-of-range value sometimes.
```

```
    ' This will fix it.
```

```
    If intI < 1 Then intI = 1
```

```
    If intI > mudtVS.StringCollection.Count Then intI = mudtVS.StringCollection.Count
```

```
15 GetRandomIndex = intI
```

```
End Function
```



```

' StringSolverx.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "StringSolver"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private mcolSV As Collection

    Private Sub Class_Initialize()

        Set mcolSV = New Collection
15
    End Sub

```

```

' SubString.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "SubString"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private mstrDelimiter As String

    Private mstrString As String

    Private mcolStr As Collection

15  Private Sub Class_Initialize()

        Set mcolStr = New Collection

    End Sub

    Public Property Let Delimiter(ByVal strNewValue As String)

20      mstrDelimiter = strNewValue

    End Property

    ' use this to convert a concatenated string to a collection
    Public Property Let StringValue(ByVal strNewValue As String)

25      mstrString = strNewValue

    End Property

    ' or use this to convert a collection to a concatenated string
30  Public Property Let StringCollection(ByVal colNewValue As Collection)

        Set mcolStr = colNewValue

    End Property

```

' converts collection into concatenated string
Public Property Get StringValue() As String

Dim varS As Variant
Dim strS As String

' build new string
For Each varS In mcolStr
strS = strS & varS & mstrDelimiter
Next varS

' trim last character
If Len(strS) > 0 Then
StringValue = left(strS, Len(strS) - 1)
End If

End Property

' converts concatenated string into a collection
Public Property Get StringCollection() As Collection

Dim colC As New Collection
Dim intI As Integer

For intI = 1 To NumSubStrings
Call colC.Add(GetSubString(intI))
Next intI

Set StringCollection = colC

End Property

' returns the number of substrings in this string
Public Property Get NumSubStrings() As Integer

Dim intD As Integer
Dim intI As Integer
Dim varS As Variant

If Len(mstrString) = 0 Then
NumSubStrings = 0
Exit Property
End If

For intI = 1 To Len(mstrString)

```
        If Mid(mstrString, intI, 1) = mstrDelimiter Then
            intD = intD + 1
        End If
    Next intI
```

```
    NumSubStrings = intD + 1
```

```
End Property
```

```
Public Sub AddSubString(ByVal strNewValue As String)
```

```
    Call mcolStr.Add(strNewValue)
```

```
End Sub
```

```
' parses the substring from the string depending on intIndex
```

```
Public Function GetSubString(ByVal intIndex As Integer) As String
```

```
    ' see if index is valid for the current string
```

```
    If NumSubStrings < intIndex Then
```

```
        GetSubString = ""
```

```
        Exit Function
```

```
    End If
```

```
    ' index into the string using delimiter
```

```
    Dim varI1 As Variant
```

```
    Dim varI2 As Variant
```

```
    Dim intCount As Integer
```

```
    varI2 = 0
```

```
    Do
```

```
        varI1 = varI2
```

```
        varI2 = InStr(varI1 + 1, mstrString, mstrDelimiter)
```

```
        intCount = intCount + 1
```

```
        If varI2 = 0 Then
```

```
            varI2 = Len(mstrString) + 1
```

```
        End If
```

```
    Loop Until intCount = intIndex
```

```
    GetSubString = Mid(mstrString, varI1 + 1, varI2 - varI1 - 1)
```

```
End Function
```



```

' Value.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1 'True
5  END
  Attribute VB_Name = "Value"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = True
  Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
  Option Explicit

  Dim mstrVariableName As String
  Dim mstrValue As String
  Dim mblnChecksum As Boolean
15  Dim mstrPrologString As String
  Dim mudtVariableType As VariableType

  Public Property Get VariableName() As String

    VariableName = mstrVariableName

20  End Property

  Public Property Let VariableName(ByVal strNewValue As String)

    mstrVariableName = strNewValue

  End Property

  Public Property Get Value() As String

25    Value = mstrValue

  End Property

  Public Property Let Value(ByVal strNewValue As String)

    mstrValue = strNewValue

30  End Property

  Public Property Get Checksum() As Boolean

```

Checksum = mblnChecksum

End Property

Public Property Let Checksum(ByVal blnNewValue As Boolean)

5 mblnChecksum = blnNewValue

End Property

Public Property Get PrologString() As String

PrologString = mstrPrologString

10 End Property

Public Property Let PrologString(ByVal strNewValue As String)

mstrPrologString = strNewValue

End Property

Public Property Get VariableType() As VariableType

15 VariableType = mudtVariableType

End Property

Public Property Let VariableType(ByVal udtNewValue As VariableType)

mudtVariableType = udtNewValue

20 End Property

```

' VarFraction.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "VarFraction"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Implements Variable

    Private mudtVar As Variable

    ' current version of data produced by this class
15  Const mintVERSIONSTAMP As Integer = 1

    Private mstrFromNum As String
    Private mstrFromDen As String
    Private mstrToNum As String
    Private mstrToDen As String
20  Private mstrByNum As String
    Private mstrByDen As String
    Private mblnMixedNumbers As Boolean
    Private mblnIsIndependent As Boolean

    Private Sub Class_Initialize()

25      Set mudtVar = New Variable

    End Sub

    Private Sub Class_Terminate()

        Set mudtVar = Nothing
30  End Sub

    ' Delegated to Class Variable
    Public Property Get Variable_Name() As String

35      Variable_Name = mudtVar.Name

```


End Property

' Delegated to Class Variable

Public Property Let Variable_Name(ByVal RHS As String)

 mudtVar.Name = RHS

5 End Property

' Delegated to Class Variable

Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

 mudtVar.Typ = udtNewValue

End Property

10 ' Delegated to Class Variable

Public Property Get Variable_Typ() As VariableType

 Variable_Typ = mudtVar.Typ

End Property

15 ' Delegated to Class Variable

Public Property Get Variable_Index() As Long

 Variable_Index = mudtVar.Index

End Property

20 ' Delegated to Class Variable

Public Property Let Variable_Index(ByVal lngNewValue As Long)

 mudtVar.Index = lngNewValue

End Property

25 ' Delegated to Class Variable

Public Property Get Variable_Enabled() As Boolean

 Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable

Public Property Let Variable_Enabled(ByVal RHS As Boolean)

 mudtVar.Enabled = RHS

End Property

5 ' Delegated to Class Variable

Public Property Get Variable_IsDirty() As Boolean

 Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable

10 Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

 mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable

15 Public Property Get Variable_Checksum() As Boolean

 Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable

20 Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

 mudtVar.Checksum = blnNewValue

End Property

Public Property Get FromNumerator() As String

25 FromNumerator = mstrFromNum

End Property

Public Property Let FromNumerator(ByVal strNewValue As String)

 mstrFromNum = strNewValue

 mudtVar.IsDirty = True

End Property

Public Property Get FromDenominator() As String

FromDenominator = mstrFromDen

End Property

5 Public Property Let FromDenominator(ByVal strNewValue As String)

mstrFromDen = strNewValue

mudtVar.IsDirty = True

End Property

Public Property Get ToNumerator() As String

10 ToNumerator = mstrToNum

End Property

Public Property Let ToNumerator(ByVal strNewValue As String)

mstrToNum = strNewValue

mudtVar.IsDirty = True

15 End Property

Public Property Get ToDenominator() As String

ToDenominator = mstrToDen

End Property

Public Property Let ToDenominator(ByVal strNewValue As String)

20 mstrToDen = strNewValue

mudtVar.IsDirty = True

End Property

Public Property Get ByNumerator() As String

ByNumerator = mstrByNum

End Property

Public Property Let ByNumerator(ByVal strNewValue As String)

mstrByNum = strNewValue
mudtVar.IsDirty = True

5 End Property

Public Property Get ByDenominator() As String

ByDenominator = mstrByDen

End Property

Public Property Let ByDenominator(ByVal strNewValue As String)

10 mstrByDen = strNewValue
mudtVar.IsDirty = True

End Property

Public Property Get MixedNumbers() As Boolean

MixedNumbers = mblnMixedNumbers

15 End Property

Public Property Let MixedNumbers(ByVal blnNewValue As Boolean)

mblnMixedNumbers = blnNewValue
mudtVar.IsDirty = True

End Property

20 Public Property Get IsIndependent() As Boolean

IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

25 mblnIsIndependent = blnNewValue
mudtVar.IsDirty = True

End Property

```
Public Sub Update(ByVal strName As String, _  
    ByVal strFromN As String, ByVal strFromD As String, _  
    ByVal strToN As String, ByVal strToD As String, _  
5    ByVal strByN As String, ByVal strByD As String, _  
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _  
    ByVal blnMixedNumber As Boolean)
```

```
    Variable_Name = strName  
10    FromNumerator = strFromN  
    FromDenominator = strFromD  
    ToNumerator = strToN  
    ToDenominator = strToD  
    ByNumerator = strByN  
15    ByDenominator = strByD  
    IsIndependent = blnIsIndependent  
    Variable_Checksum = blnChecksum  
    MixedNumbers = blnMixedNumber
```

```
20 End Sub
```

```
Public Function Variable_PrologFormat() As String
```

```
    Dim str1 As String
```

```
25    If mblnIsIndependent Then
```

```
        str1 = "fraction(" & mudtVar.Name & "),offgrid(" & _  
            mudtVar.Name & "),[" & _  
            mstrFromNum & "/" & mstrFromDen & "<=" & _  
            mudtVar.Name & "<=" & mstrToNum & "/" & _  
30        mstrToDen & " step " & mstrByNum & "/" & mstrByDen & "]"
```

```
    Else
```

```
        str1 = "fraction(" & mudtVar.Name & ")"
```

```
    End If
```

```
35    Variable_PrologFormat = str1
```

```
End Function
```

```
Public Function Variable_ScreenFormat() As String
```

```
    Dim str1 As String
```

```
40    Dim strOpt As String
```

```

    If mudtVar.Checksum Then
        strOpt = "(C,"
    Else
5       strOpt = "(c,"
    End If

    If mblnMixedNumbers Then
        strOpt = strOpt & "M),"
10    Else
        strOpt = strOpt & "m),"
    End If

    If mblnIsIndependent Then
15        str1 = mudtVar.Name & strOpt & ": Fraction, " & _
            mstrFromNum & "/" & mstrFromDen & " to " & _
            mstrToNum & "/" & mstrToDen & " by " & _
            mstrByNum & "/" & mstrByDen
    Else
20        str1 = mudtVar.Name & strOpt & ": Fraction"
    End If

    Variable_ScreenFormat = str1

End Function

25 Public Property Get Variable_ReadType(udtFile As File) As VariableType

    Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

    Dim vField As Variant

30    Call udtFile.ReadField(vField) ' reads version stamp
    Call udtFile.ReadField(vField)
    mudtVar.Name = vField

    Call udtFile.ReadField(vField)
35    mudtVar.Enabled = vField

    Call udtFile.ReadField(vField)
    mudtVar.Checksum = vField

```

```

    Call udtFile.ReadField(vField)
    IsIndependent = vField

5    Call udtFile.ReadField(vField)
    FromNumerator = vField

    Call udtFile.ReadField(vField)
    FromDenominator = vField

10    Call udtFile.ReadField(vField)
    ToNumerator = vField

    Call udtFile.ReadField(vField)
    ToDenominator = vField

15    Call udtFile.ReadField(vField)
    ByNumerator = vField

    Call udtFile.ReadField(vField)
    ByDenominator = vField

    Call udtFile.ReadField(vField)
    MixedNumbers = vField

25 End Sub

Public Sub Variable_ WriteObjectData(udtFile As File)

    Dim udtType As VariableType

    udtType = vtFraction
30    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
35    Call udtFile.WriteField(IsIndependent)
    Call udtFile.WriteField(FromNumerator)
    Call udtFile.WriteField(FromDenominator)
    Call udtFile.WriteField(ToNumerator)
    Call udtFile.WriteField(ToDenominator)
40    Call udtFile.WriteField(ByNumerator)
    Call udtFile.WriteField(ByDenominator)
    Call udtFile.WriteField(MixedNumbers)

```

```
    mudtVar.IsDirty = False
```

```
End Sub
```

```
' makes a copy of this object
```

```
5 Public Function Variable_Copy() As Variable
```

```
    Dim udtVF As New VarFraction
```

```
    Dim udtV As Variable
```

```
    Set udtV = udtVF
```

```
10
```

```
    udtV.Name = mudtVar.Name
```

```
    udtV.Enabled = mudtVar.Index
```

```
    udtV.IsDirty = mudtVar.IsDirty
```

```
    udtV.Checksum = mudtVar.Checksum
```

```
15
```

```
    udtVF.FromNumerator = FromNumerator
```

```
    udtVF.FromDenominator = FromDenominator
```

```
    udtVF.ByNumerator = ByNumerator
```

```
    udtVF.ByDenominator = ByDenominator
```

```
20
```

```
    udtVF.ToNumerator = ToNumerator
```

```
    udtVF.ToDenominator = ToDenominator
```

```
    udtVF.IsIndependent = IsIndependent
```

```
    udtVF.MixedNumbers = MixedNumbers
```

```
25
```

```
    Set Variable_Copy = udtV
```

```
End Function
```



```

' Variable.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = 0 'False
5    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
10    Attribute VB_Name = "Variable"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
    Attribute VB_Exposed = False
15    Attribute VB_Ext_KEY = "SavedWithClassBuilder" ,"Yes"
    Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
    Option Explicit

    Private mstrName As String
    Private mudtType As VariableType
20    Private mlngIndex As Long
    Private mblnEnabled As Boolean
    Private mblnIsDirty As Boolean
    Private mblnChecksum As Boolean

    Public Enum VariableType
25        vtInteger = 0
        vtReal = 1
        vtFraction = 2
        vtString = 3
        vtUntyped = 4
30    End Enum

    Public Property Get name() As String

        name = mstrName

    End Property

    Public Property Let name(ByVal strNewValue As String)

35    If mstrName <> strNewValue Then
        mstrName = strNewValue
        mblnIsDirty = True

```

End If

End Property

Public Property Get Typ() As VariableType

Typ = mudtType

End Property

Public Property Let Typ(ByVal udtNewValue As VariableType)

If mudtType <> udtNewValue Then

mudtType = udtNewValue

mblnIsDirty = True

End If

End Property

Public Property Get index() As Long

index = mlngIndex

End Property

Public Property Let index(ByVal lngNewValue As Long)

If mlngIndex <> lngNewValue Then

mlngIndex = lngNewValue

mblnIsDirty = True

End If

End Property

Public Property Get Enabled() As Boolean

Enabled = mblnEnabled

End Property

Public Property Let Enabled(ByVal blnNewValue As Boolean)

If mblnEnabled <> blnNewValue Then

mblnEnabled = blnNewValue

mblnIsDirty = True

End If

End Property

Public Property Let IsDirty(ByVal blnNewValue As Boolean)

5 mblnIsDirty = blnNewValue

End Property

Public Property Get IsDirty() As Boolean

IsDirty = mblnIsDirty

10

End Property

Public Property Let Checksum(ByVal blnNewValue As Boolean)

If mblnChecksum <> blnNewValue Then

 mblnChecksum = blnNewValue

 mblnIsDirty = True

End If

15

End Property

Public Property Get Checksum() As Boolean

Checksum = mblnChecksum

20

End Property

' implemented in the subclasses of Variable

Public Function PrologFormat() As String

25

End Function

' implemented in the subclasses of Variable

Public Function ScreenFormat() As String

30

End Function

' implemented in the subclasses of Variable

```
Public Sub ReadObjectData(udtFile As File)
```

```
End Sub
```

```
' implemented in the subclasses of Variable
```

```
Public Sub WriteObjectData(udtFile As File)
```

```
5 End Sub
```

```
Public Property Get ReadType(udtFile As File) As VariableType
```

```
Dim udtType As VariableType
```

```
Call udtFile.ReadField(udtType)
```

```
10 ReadType = udtType
```

```
End Property
```

```
' implemented in the subclasses of Variable
```

```
Public Function Copy() As Variable
```

```
15 End Function
```

```

' VarInteger.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "VarInteger"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Implements Variable

    Private mudtVar As Variable

    ' current version of data produced by this class
15  Const mintVERSIONSTAMP As Integer = 1

    Private mstrFrom As String
    Private mstrTo As String
    Private mstrBy As String
    Private mblnIsIndependent As Boolean

20  Private Sub Class_Initialize()

        Set mudtVar = New Variable

    End Sub

    Private Sub Class_Terminate()

25        Set mudtVar = Nothing

    End Sub

    ' Delegated to Class Variable
    Public Property Get Variable_Name() As String

30        Variable_Name = mudtVar.Name

    End Property

    ' Delegated to Class Variable

```

```
Public Property Let Variable_Name(ByVal RHS As String)
```

```
    mudtVar.Name = RHS
```

```
End Property
```

```
' Delegated to Class Variable
```

```
5 Public Property Get Variable_Typ() As VariableType
```

```
    Variable_Typ = mudtVar.Typ
```

```
End Property
```

```
' Delegated to Class Variable
```

```
10 Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)
```

```
    mudtVar.Typ = udtNewValue
```

```
End Property
```

```
' Delegated to Class Variable
```

```
Public Property Get Variable_Index() As Long
```

```
15 Variable_Index = mudtVar.Index
```

```
End Property
```

```
' Delegated to Class Variable
```

```
Public Property Let Variable_Index(ByVal lngNewValue As Long)
```

```
20 mudtVar.Index = lngNewValue
```

```
End Property
```

```
' Delegated to Class Variable
```

```
Public Property Get Variable_Enabled() As Boolean
```

```
25 Variable_Enabled = mudtVar.Enabled
```

```
End Property
```

```
' Delegated to Class Variable
```

```
Public Property Let Variable_Enabled(ByVal RHS As Boolean)
```

```
    mudtVar.Enabled = RHS
```

End Property

' Delegated to Class Variable

Public Property Get Variable_IsDirty() As Boolean

5 Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable

Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

 mudtVar.IsDirty = RHS

10

End Property

' Delegated to Class Variable

Public Property Get Variable_Checksum() As Boolean

 Variable_Checksum = mudtVar.Checksum

15

End Property

' Delegated to Class Variable

Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

 mudtVar.Checksum = blnNewValue

20

End Property

Public Property Get From() As String

 From = mstrFrom

End Property

25 Public Property Let From(ByVal strNewValue As String)

 If mstrFrom <> strNewValue Then

 mstrFrom = strNewValue

 mudtVar.IsDirty = True

 End If

30 End Property

Public Property Get Too() As String

 Too = mstrTo

End Property

Public Property Let Too(ByVal strNewValue As String)

5 If mstrTo <> strNewValue Then
 mstrTo = strNewValue
 mudtVar.IsDirty = True
 End If

End Property

10 Public Property Get By() As String

 By = mstrBy

End Property

Public Property Let By(ByVal strNewValue As String)

15 If mstrBy <> strNewValue Then
 mstrBy = strNewValue
 mudtVar.IsDirty = True
 End If

End Property

Public Property Get IsIndependent() As Boolean

20 IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

25 If mblnIsIndependent <> blnNewValue Then
 mblnIsIndependent = blnNewValue
 mudtVar.IsDirty = True
 End If

End Property


```

Public Sub Update(ByVal strName As String, _
    ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
    ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean)

5   Variable_Name = strName
    From = strFrom
    Too = strTo
    By = strBy
    IsIndependent = blnIsIndependent
10   Variable_Checksum = blnChecksum

End Sub

Public Function Variable_PrologFormat() As String

    Dim str1 As String
15   If mblnIsIndependent Then
        str1 = "int(" & mudtVar.Name & "),[" & mstrFrom & "<=" & _
            mudtVar.Name & "<=" & mstrTo & " step " & mstrBy & "]"
    Else
20   str1 = "int(" & mudtVar.Name & ")"
    End If

    Variable_PrologFormat = str1

End Function

25   Public Function Variable_ScreenFormat() As String

        Dim str1 As String
        Dim strT As String
        Dim strOpt As String

30   If mudtVar.Checksum Then
        strOpt = "(C)"
    Else
        strOpt = "(c)"
    End If

35   If mblnIsIndependent Then
        str1 = mudtVar.Name & strOpt & ": Int, " & mstrFrom & " to " & _
            mstrTo & " by " & mstrBy
    Else
40   str1 = mudtVar.Name & strOpt & ": Int"

```

End If

Variable_ScreenFormat = str1

End Function

5 Public Property Get Variable_ReadType(udtFile As File) As VariableType

Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

Dim vField As Variant

10 Call udtFile.ReadField(vField) ' reads version stamp

Call udtFile.ReadField(vField)
mudtVar.Name = vField

15 Call udtFile.ReadField(vField)
mudtVar.Enabled = vField

Call udtFile.ReadField(vField)
mudtVar.Checksum = vField

20 Call udtFile.ReadField(vField)
From = vField

25 Call udtFile.ReadField(vField)
Too = vField

Call udtFile.ReadField(vField)
By = vField

30 Call udtFile.ReadField(vField)
IsIndependent = vField

End Sub

Public Sub Variable_WriteObjectData(udtFile As File)

Dim udtType As VariableType

35

```

    udtType = vtInteger
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
5    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
    Call udtFile.WriteField(From)
    Call udtFile.WriteField(Too)
    Call udtFile.WriteField(By)
10   Call udtFile.WriteField(IsIndependent)

```

```

    mudtVar.IsDirty = False

```

```

End Sub

```

```

' makes a copy of this object
15 Public Function Variable_Copy() As Variable

```

```

    Dim udtVI As New VarInteger
    Dim udtV As Variable

```

```

    Set udtV = udtVI

```

```

20   udtV.Name = mudtVar.Name
    udtV.Type = vtInteger
    udtV.Enabled = mudtVar.Index
    udtV.IsDirty = mudtVar.IsDirty
25   udtV.Checksum = mudtVar.Checksum

```

```

    udtVI.From = From
    udtVI.Too = Too
    udtVI.By = By
30   udtVI.IsIndependent = IsIndependent

```

```

    Set Variable_Copy = udtV

```

```

End Function

```

```

' VarReal.cls
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1 'True
5  END
  Attribute VB_Name = "VarReal"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = True
  Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
  Option Explicit

  Implements Variable

  Private mudtVar As Variable

  ' current version of data produced by this class
15  Const mintVERSIONSTAMP As Integer = 2

  Private mstrFrom As String
  Private mstrTo As String
  Private mstrBy As String
  Private mblnTrailingZeros As Boolean
20  Private mstrPrecision As String
  Private mblnIsIndependent As Boolean
  Private mblnIsOnGrid As Boolean

  Private Sub Class_Initialize()

    Set mudtVar = New Variable
25  End Sub

  Private Sub Class_Terminate()

    Set mudtVar = Nothing

30  End Sub

  ' Delegated to Class Variable
  Public Property Get Variable_Name() As String

    Variable_Name = mudtVar.Name

```

End Property

' Delegated to Class Variable

Public Property Let Variable_Name(ByVal RHS As String)

 mudtVar.Name = RHS

5 End Property

' Delegated to Class Variable

Public Property Get Variable_Typ() As VariableType

 Variable_Typ = mudtVar.Typ

10 End Property

' Delegated to Class Variable

Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

 mudtVar.Typ = udtNewValue

15 End Property

' Delegated to Class Variable

Public Property Get Variable_Enabled() As Boolean

 Variable_Enabled = mudtVar.Enabled

End Property

20 ' Delegated to Class Variable

Public Property Let Variable_Enabled(ByVal RHS As Boolean)

 mudtVar.Enabled = RHS

End Property

25 ' Delegated to Class Variable

Public Property Get Variable_Index() As Long

 Variable_Index = mudtVar.Index

End Property

30 ' Delegated to Class Variable

```
Public Property Let Variable_Index(ByVal lngNewValue As Long)
```

```
    mudtVar.Index = lngNewValue
```

```
End Property
```

```
5    ' Delegated to Class Variable  
Public Property Get Variable_IsDirty() As Boolean
```

```
    Variable_IsDirty = mudtVar.IsDirty
```

```
End Property
```

```
10    ' Delegated to Class Variable  
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)
```

```
    mudtVar.IsDirty = RHS
```

```
End Property
```

```
15    ' Delegated to Class Variable  
Public Property Get Variable_Checksum() As Boolean
```

```
    Variable_Checksum = mudtVar.Checksum
```

```
End Property
```

```
20    ' Delegated to Class Variable  
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)
```

```
    mudtVar.Checksum = blnNewValue
```

```
End Property
```

```
Public Property Get From() As String
```

```
25    From = mstrFrom
```

```
End Property
```

```
Public Property Let From(ByVal strNewValue As String)
```

```
    If mstrFrom <> strNewValue Then
```

```
        mstrFrom = strNewValue
```

```
30    mudtVar.IsDirty = True
```

End If

End Property

Public Property Get Too() As String

Too = mstrTo

5 End Property

Public Property Let Too(ByVal strNewValue As String)

If mstrTo <> strNewValue Then

mstrTo = strNewValue

mudtVar.IsDirty = True

10 End If

End Property

Public Property Get By() As String

By = mstrBy

End Property

15 Public Property Let By(ByVal strNewValue As String)

If mstrBy <> strNewValue Then

mstrBy = strNewValue

mudtVar.IsDirty = True

End If

20 End Property

Public Property Get TrailingZeros() As Boolean

TrailingZeros = mblnTrailingZeros

End Property

Public Property Let TrailingZeros(ByVal blnNewValue As Boolean)

25 If mblnTrailingZeros <> blnNewValue Then

mblnTrailingZeros = blnNewValue

mudtVar.IsDirty = True

End If

End Property

Public Property Get IsOnGrid() As Boolean

IsOnGrid = mblnIsOnGrid

5 End Property

Public Property Let IsOnGrid(ByVal blnNewValue As Boolean)

If mblnIsOnGrid <> blnNewValue Then

mblnIsOnGrid = blnNewValue

mutdVar.IsDirty = True

10 End If

End Property

Public Property Get Precision() As String

Precision = mstrPrecision

End Property

15 Public Property Let Precision(ByVal strNewValue As String)

If mstrPrecision <> strNewValue Then

mstrPrecision = strNewValue

mutdVar.IsDirty = True

End If

20 End Property

Public Property Get DecimalPlaces() As Integer

If InStr(1, mstrPrecision, ".") = 0 Then

DecimalPlaces = 0

Else

25 DecimalPlaces = Len(mstrPrecision) - 1

End If

End Property

Public Property Get IsIndependent() As Boolean

IsIndependent = mblnIsIndependent

End Property

Public Property Let IsIndependent(ByVal blnNewValue As Boolean)

5 If mblnIsIndependent <> blnNewValue Then
 mblnIsIndependent = blnNewValue
 mudtVar.IsDirty = True
 End If

End Property

10 Public Sub Update(ByVal strName As String, _
 ByVal strFrom As String, ByVal strTo As String, ByVal strBy As String, _
 ByVal blnIsIndependent As Boolean, ByVal blnChecksum As Boolean, _
 ByVal blnTrailingZeros As Boolean, _
 ByVal strPrecision As String, ByVal blnIsOnGrid As Boolean)

15 Variable_Name = strName
 From = strFrom
 Too = strTo
 By = strBy
 IsIndependent = blnIsIndependent
20 Variable_Checksum = blnChecksum
 TrailingZeros = blnTrailingZeros
 Precision = strPrecision
 IsOnGrid = blnIsOnGrid

25 End Sub

Public Function Variable_PrologFormat() As String

Dim str1 As String

30 If mblnIsIndependent Then
 str1 = "real({" & mudtVar.Name & "," & mstrPrecision & "}),[" & _
 & mstrFrom & "<=" & mudtVar.Name & "<=" & mstrTo & " step " & _
 mstrBy & "]"

Else

35 str1 = "real(" & mudtVar.Name & ")"
End If

If Not mblnIsOnGrid Then

str1 = str1 & ",offgrid(" & mudtVar.Name & ")"

End If

Variable_PrologFormat = str1

End Function

5 Public Function Variable_ScreenFormat() As String

Dim str1 As String

Dim strOpt As String

10 If mudtVar.Checksum Then

strOpt = "(C,"

Else

strOpt = "(c,"

End If

15

If mblnTrailingZeros Then

strOpt = strOpt & "T,"

Else

strOpt = strOpt & "t,"

20

End If

If mblnIsOnGrid Then

strOpt = strOpt & "G,"

Else

25

strOpt = strOpt & "g,"

End If

strOpt = strOpt & mstrPrecision & ")"

30

If mblnIsIndependent Then

str1 = mudtVar.Name & strOpt & ": Real, " & mstrFrom & " to " & _
mstrTo & " by " & mstrBy

Else

str1 = mudtVar.Name & strOpt & ": Real"

35

End If

Variable_ScreenFormat = str1

End Function

Public Property Get Variable_ReadType(udtFile As File) As VariableType

40 Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

Dim vField As Variant

Dim intVersion As Integer

5

Call udtFile.ReadField(vField) ' reads version stamp
intVersion = vField

10

Call udtFile.ReadField(vField)
mudtVar.Name = vField

Call udtFile.ReadField(vField)
mudtVar.Enabled = vField

15

Call udtFile.ReadField(vField)
mudtVar.Checksum = vField

20

Call udtFile.ReadField(vField)
From = vField

Call udtFile.ReadField(vField)
Too = vField

25

Call udtFile.ReadField(vField)
By = vField

Call udtFile.ReadField(vField)
TrailingZeros = vField

30

Call udtFile.ReadField(vField)
Precision = vField

Call udtFile.ReadField(vField)
IsIndependent = vField

35

If intVersion < 2 Then ' this field is new to version 2 of VarReal
IsOnGrid = True

Else

Call udtFile.ReadField(vField)
IsOnGrid = vField

40

End If

End Sub

```
Public Sub Variable_WriteObjectData(udtFile As File)
```

```
    Dim udtType As VariableType
```

```
    udtType = vtReal
```

```
5    Call udtFile.WriteField(udtType)
```

```
    Call udtFile.WriteField(mintVERSIONSTAMP)
```

```
    Call udtFile.WriteField(mudtVar.Name)
```

```
    Call udtFile.WriteField(mudtVar.Enabled)
```

```
    Call udtFile.WriteField(mudtVar.Checksum)
```

```
10    Call udtFile.WriteField(From)
```

```
    Call udtFile.WriteField(Too)
```

```
    Call udtFile.WriteField(By)
```

```
    Call udtFile.WriteField(TrailingZeros)
```

```
    Call udtFile.WriteField(Precision)
```

```
15    Call udtFile.WriteField(IsIndependent)
```

```
    Call udtFile.WriteField(IsOnGrid)
```

```
    mudtVar.IsDirty = False
```

```
End Sub
```

```
20 ' makes a copy of this object
```

```
Public Function Variable_Copy() As Variable
```

```
    Dim udtVR As New VarReal
```

```
    Dim udtV As Variable
```

```
25    Set udtV = udtVR
```

```
    udtV.Name = mudtVar.Name
```

```
    udtV.Type = vtReal
```

```
    udtV.Enabled = mudtVar.Index
```

```
30    udtV.IsDirty = mudtVar.IsDirty
```

```
    udtV.Checksum = mudtVar.Checksum
```

```
    udtVR.From = From
```

```
    udtVR.Too = Too
```

```
35    udtVR.By = By
```

```
    udtVR.Precision = Precision
```

```
    udtVR.TrailingZeros = TrailingZeros
```

```
    udtVR.IsIndependent = IsIndependent
```

```
    udtVR.IsOnGrid = IsOnGrid
```

```
40    Set Variable_Copy = udtV
```

End Function

```

' VarString.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "VarString"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Implements Variable

    Private mudtVar As Variable

    ' current version of data produced by this class
15  Const mintVERSIONSTAMP As Integer = 1

    Private mstrDelimiter As String

    Private mblnIsIndexed As Boolean

    Private mcolString As New Collection

    Private Sub Class_Initialize()
20      Set mudtVar = New Variable
    End Sub

    Private Sub Class_Terminate()

        Set mudtVar = Nothing
25  End Sub

    ' Delegated to Class Variable
    Public Property Get Variable_Name() As String
30      Variable_Name = mudtVar.Name
    End Property

```

' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

 mudtVar.Name = RHS

End Property

5 ' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

 Variable_Typ = mudtVar.Typ

End Property

10 ' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

 mudtVar.Typ = udtNewValue

End Property

15 ' Delegated to Class Variable
Public Property Get Variable_Index() As Long

 Variable_Index = mudtVar.Index

End Property

20 ' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

 mudtVar.Index = lngNewValue

End Property

25 ' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

 Variable_Enabled = mudtVar.Enabled

End Property

' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

```

    mudtVar.Enabled = RHS

End Property

' Delegated to Class Variable
5 Public Property Get Variable_IsDirty() As Boolean

    Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

10     mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
Public Property Get Variable_Checksum() As Boolean

15     Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

20     mudtVar.Checksum = blnNewValue

End Property

Public Property Get Delimiter() As String

    Delimiter = mstrDelimiter

25 End Property

Public Property Let Delimiter(ByVal strNewValue As String)

    If mstrDelimiter <> strNewValue Then
        mstrDelimiter = strNewValue
        mudtVar.IsDirty = True
    End If

30 End If

```


End Property

Public Property Get IsIndexed() As Boolean

IsIndexed = mblnIsIndexed

5 End Property

Public Property Let IsIndexed(ByVal blnNewValue As Boolean)

mblnIsIndexed = blnNewValue

End Property

10 Public Property Get StringCollection() As Collection

Set StringCollection = mcolString

End Property

Public Property Let StringCollection(ByVal colNewValue As Collection)

15 Dim intIndex As Integer

If mcolString.Count <> colNewValue.Count Then

Set mcolString = colNewValue

mudtVar.IsDirty = True

20 Exit Property

End If

For intIndex = 1 To mcolString.Count

If mcolString.Item(intIndex) <> colNewValue.Item(intIndex) Then

25 Set mcolString = colNewValue

mudtVar.IsDirty = True

Exit Property

End If

Next intIndex

30

End Property

' returns the largest number of delimited substrings in the string collection

Public Property Get NumIndices() As Integer

Dim intD As Integer

35 Dim intHiD As Integer

```

Dim intI As Integer
Dim varS As Variant
Dim udtSubStr As New SubString

5   ' if there are no strings in the collection
    If mcolString.Count = 0 Then
        NumIndices = 1
        Exit Property
    End If

10   udtSubStr.Delimiter = mstrDelimiter

    For Each varS In mcolString
        udtSubStr.StringValue = varS
15   intD = udtSubStr.NumSubStrings
        If intD > intHiD Then
            intHiD = intD
        End If
    Next varS

20   NumIndices = intHiD

End Property

Public Function Variable_PrologFormat() As String

25   Variable_PrologFormat = ""

End Function

Public Function Variable_ScreenFormat() As String

30   Dim str1 As String
    Dim strS As String
    Dim intIndex As Integer
    Dim strOpt As String

    If mudtVar.Checksum Then
35   strOpt = "(C,"
    Else
        strOpt = "(c,"
    End If

40   strOpt = strOpt & Str(NumIndices) & "," & mstrDelimiter & ")"

```

```
For intIndex = 1 To 3
```

```
    If mcolString.Count >= intIndex Then  
        strS = strS & mcolString.Item(intIndex)  
5      If mcolString.Count > intIndex Then  
        strS = strS & ","  
        End If  
    End If
```

```
10  Next intIndex
```

```
    If mcolString.Count > 3 Then  
        strS = strS & "..."  
    End If
```

```
15  str1 = mudtVar.Name & strOpt & ": String, in [" & strS & "]"
```

```
    Variable_ScreenFormat = str1
```

```
End Function
```

```
20  Public Property Get Variable_ReadType(udtFile As File) As VariableType
```

```
    Variable_ReadType = mudtVar.ReadType(udtFile)
```

```
End Property
```

```
Public Sub Variable_ReadObjectData(udtFile As File)
```

```
25  Dim vField As Variant  
    Dim intCount As Integer
```

```
    Call udtFile.ReadField(vField) ' reads version stamp  
    Call udtFile.ReadField(vField)  
    mudtVar.Name = vField
```

```
30  Call udtFile.ReadField(vField)  
    mudtVar.Enabled = vField
```

```
35  Call udtFile.ReadField(vField)  
    mudtVar.Checksum = vField
```

```
    Call udtFile.ReadField(vField)  
    mstrDelimiter = vField
```

```
Call udtFile.ReadField(vField)
mblnIsIndexed = vField
```

```
Call udtFile.ReadField(vField)
intColor = vField
```

```
Dim intI As Integer
```

```
' read in the strings
For intI = 1 To intCount
```

```
    Call udtFile.ReadField(vField)
    Call mcolString.Add(vField)
```

```
Next intI
```

```
End Sub
```

```
Public Sub Variable_WriteObjectData(udtFile As File)
```

```
    Dim udtType As VariableType
```

```
    udtType = vtString
    Call udtFile.WriteField(udtType)
    Call udtFile.WriteField(mintVERSIONSTAMP)
    Call udtFile.WriteField(mudtVar.Name)
    Call udtFile.WriteField(mudtVar.Enabled)
    Call udtFile.WriteField(mudtVar.Checksum)
    Call udtFile.WriteField(mstrDelimiter)
    Call udtFile.WriteField(mblnIsIndexed)
```

```
    Dim intCount As Integer
```

```
    intCount = mcolString.Count
    Call udtFile.WriteField(intCount)
```

```
    Dim intI As Integer
```

```
' write out the strings
For intI = 1 To mcolString.Count
    Call udtFile.WriteField(mcolString.Item(intI))
Next intI
```

```
mudtVar.IsDirty = False
```

End Sub

' makes a copy of this object

Public Function Variable_Copy() As Variable

Dim udtVS As New VarString

Dim udtV As Variable

Dim varS As Variant

Set udtV = udtVS

udtV.Name = mudtVar.Name

udtV.Typ = vtString

udtV.Enabled = mudtVar.Index

udtV.IsDirty = mudtVar.IsDirty

udtV.Checksum = mudtVar.Checksum

udtVS.Delimiter = Delimiter

udtVS.IsIndexed = IsIndexed

Set Variable_Copy = udtV

For Each varS In mcolString

Call udtVS.StringCollection.Add(varS)

Next varS

End Function

' VarUntyped.cls

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB_Name = "VarUntyped"

Attribute VB_GlobalNameSpace = False

Attribute VB_Creatable = True

Attribute VB_PredeclaredId = False

Attribute VB_Exposed = False

Option Explicit

Implements Variable

```

Private mudtVar As Variable

' current version of data produced by this class
Const mintVERSIONSTAMP As Integer = 1

Private Sub Class_Initialize()

5     Set mudtVar = New Variable

End Sub

Private Sub Class_Terminate()

    Set mudtVar = Nothing
10 End Sub

' Delegated to Class Variable
Public Property Get Variable_Name() As String

15     Variable_Name = mudtVar.Name

End Property

' Delegated to Class Variable
Public Property Let Variable_Name(ByVal RHS As String)

    mudtVar.Name = RHS
20 End Property

' Delegated to Class Variable
Public Property Get Variable_Typ() As VariableType

    Variable_Typ = mudtVar.Typ
25 End Property

' Delegated to Class Variable
Public Property Let Variable_Typ(ByVal udtNewValue As VariableType)

    mudtVar.Typ = udtNewValue

End Property

```

```

' Delegated to Class Variable
Public Property Get Variable_Index() As Long

    Variable_Index = mudtVar.Index

5    End Property

' Delegated to Class Variable
Public Property Let Variable_Index(ByVal lngNewValue As Long)

    mudtVar.Index = lngNewValue

10   End Property

' Delegated to Class Variable
Public Property Get Variable_Enabled() As Boolean

    Variable_Enabled = mudtVar.Enabled

End Property

15   ' Delegated to Class Variable
Public Property Let Variable_Enabled(ByVal RHS As Boolean)

    mudtVar.Enabled = RHS

End Property

20   ' Delegated to Class Variable
Public Property Get Variable_IsDirty() As Boolean

    Variable_IsDirty = mudtVar.IsDirty

End Property

' Delegated to Class Variable
25   Public Property Let Variable_IsDirty(ByVal RHS As Boolean)

    mudtVar.IsDirty = RHS

End Property

' Delegated to Class Variable
30   Public Property Get Variable_Checksum() As Boolean

```

```

    Variable_Checksum = mudtVar.Checksum

End Property

' Delegated to Class Variable
5  Public Property Let Variable_Checksum(ByVal blnNewValue As Boolean)

    mudtVar.Checksum = blnNewValue

End Property

Public Function Variable_PrologFormat() As String
10  Variable_PrologFormat = ""

End Function

Public Function Variable_ScreenFormat() As String

15  Dim str1 As String
    Dim strS As String
    Dim intIndex As Integer
    Dim strOpt As String

20  If mudtVar.Checksum Then
        strOpt = "(C)"
    Else
        strOpt = "(c)"
    End If

25  str1 = mudtVar.Name & strOpt & ": Untyped"

    Variable_ScreenFormat = str1

End Function

30  Public Property Get Variable_ReadType(udtFile As File) As VariableType

    Variable_ReadType = mudtVar.ReadType(udtFile)

End Property

Public Sub Variable_ReadObjectData(udtFile As File)

    Dim vField As Variant

```



```
Dim intCount As Integer
```

```
Call udtFile.ReadField(vField) ' reads version stamp  
Call udtFile.ReadField(vField)  
5 mudtVar.Name = vField
```

```
Call udtFile.ReadField(vField)  
mudtVar.Enabled = vField
```

```
10 Call udtFile.ReadField(vField)  
mudtVar.Checksum = vField
```

```
End Sub
```

```
Public Sub Variable_WriteObjectData(udtFile As File)
```

```
15 Dim udtType As VariableType
```

```
udtType = vtUntyped  
Call udtFile.WriteField(udtType)  
Call udtFile.WriteField(mintVERSIONSTAMP)  
20 Call udtFile.WriteField(mudtVar.Name)  
Call udtFile.WriteField(mudtVar.Enabled)  
Call udtFile.WriteField(mudtVar.Checksum)
```

```
mudtVar.IsDirty = False
```

```
25 End Sub
```

```
' makes a copy of this object
```

```
Public Function Variable_Copy() As Variable
```

```
Dim udtV As New Variable
```

```
30 udtV.Name = mudtVar.Name  
udtV.Type = vtUntyped  
udtV.Enabled = mudtVar.Index  
udtV.IsDirty = mudtVar.IsDirty  
udtV.Checksum = mudtVar.Checksum
```

```
35 Set Variable_Copy = udtV
```

```
End Function
```



```
' Win32API.cls
VERSION 1.0 CLASS
BEGIN
```

```
MultiUse = -1 'True
```

```
END
```

```
Attribute VB_Name = "Win32API"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = True
```

```
Attribute VB_PredeclaredId = False
```

```
Attribute VB_Exposed = False
```

```
' used for making calls to the Win32 API
```

```
Option Explicit
```

```
Private Type FILETIME
```

```
dwLowDateTime As Long
```

```
dwHighDateTime As Long
```

```
End Type
```

```
Private Const MAX_PATH = 260
```

```
Private Type WIN32_FIND_DATA
```

```
dwFileAttributes As Long
```

```
ftCreationTime As FILETIME
```

```
ftLastAccessTime As FILETIME
```

```
ftLastWriteTime As FILETIME
```

```
nFileSizeHigh As Long
```

```
nFileSizeLow As Long
```

```
dwReserved0 As Long
```

```
dwReserved1 As Long
```

```
cFileName As String * MAX_PATH
```

```
cAlternate As String * 14
```

```
End Type
```

```
Private Const INVALID_HANDLE_VALUE = -1
```

```
Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
    (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
```

```
Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" _
    (ByVal hFileName As Long, lpFindFileData As WIN32_FIND_DATA) As Long
```

```
Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long
```

```
Private Declare Function GetCurrentDirectory Lib "kernel32" _
```

```
Alias "GetCurrentDirectoryA" (ByVal nBufferLength As Long, _  
ByVal lpBuffer As String) As Long
```

```
Private Declare Function SendMessageLong Lib "user32" Alias "SendMessageA" _  
5 (ByVal hwnd As Long, _  
ByVal Msg As Long, _  
ByVal wParam As Long, _  
ByVal lParam As Long) As Long
```

```
Private Declare Function SystemParametersInfo Lib "user32" _  
10 Alias "SystemParametersInfoA" (ByVal uAction As Long, _  
ByVal uParam As Long, ByRef lpvParam As Any, _  
ByVal fuWinIni As Long) As Long
```

```
Private Const SPI_GETDRAGFULLWINDOWS = 38  
Private Const SPI_SETDRAGFULLWINDOWS = 37  
Private Const SPIF_SENDWININICHANGE = 2
```

```
Public Function IsFullWindowDragOn() As Boolean
```

```
Dim result As Long
```

```
'Call API and check for successful call.
```

```
If SystemParametersInfo(SPI_GETDRAGFULLWINDOWS, 0&, result, 0&) <> 0 Then
```

```
'Feature supported now check value of result.
```

```
If result = 0 Then
```

```
IsFullWindowDragOn = False
```

```
Else
```

```
IsFullWindowDragOn = True
```

```
End If
```

```
'Call failed, feature not supported.
```

```
Else
```

```
IsFullWindowDragOn = False
```

```
End If
```

```
End Function
```

```
Public Sub TurnOffFullWindowDrag()
```

```
Dim result As Long
```

```
35 result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 0&, _  
ByVal vbNullString, SPIF_SENDWININICHANGE)
```

```
End Sub
```

```
Public Sub TurnOnFullWindowDrag()
```

```
    Dim result As Long
```

```
    result = SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, 1&, _  
    ByVal vbNullString, SPIF_SENDWININICHANGE)
```

```
End Sub
```

```
' returns true if strFN exists
```

```
Public Function FileExists(ByVal strFN) As Boolean
```

```
    Dim lngHandle As Long
```

```
    Dim w32FindData As WIN32_FIND_DATA
```

```
    lngHandle = FindFirstFile(strFN, w32FindData)
```

```
    If lngHandle = INVALID_HANDLE_VALUE Then
```

```
        FileExists = False
```

```
    Else
```

```
        FileExists = True
```

```
        Call FindClose(lngHandle)
```

```
    End If
```

```
End Function
```

```
' returns a collection of file names that satisfy strMask. The path seems to  
' disappear from the returned file names.
```

```
Public Function FindAllFiles(ByVal strMask As String) As Collection
```

```
    Dim lngHandle As Long
```

```
    Dim lngRet As Long
```

```
    Dim w32FindData As WIN32_FIND_DATA
```

```
    Dim strFN As String
```

```
    Dim varI As Variant
```

```
    Dim colFNs As New Collection
```

```
    lngHandle = FindFirstFile(strMask, w32FindData)
```

```
    If lngHandle = INVALID_HANDLE_VALUE Then
```

```
        Exit Function
```

```
    End If
```

```
    Do
```

```

        strFN = TrimAtFirstNull(w32FindData.cFileName)
        Call colFNs.Add(strFN) ' add to the collection

    Loop Until FindNextFile(lngHandle, w32FindData) = 0
5
    Set FindAllFiles = colFNs

End Function

' returns the current directory
10 Public Function CurrentDirectory() As String

    Dim strBuf As String
    Dim lngRet As Long
    Dim varI As Variant

15    strBuf = Space(300)
    lngRet = GetCurrentDirectory(300, strBuf)
    CurrentDirectory = TrimAtFirstNull(strBuf)

End Function

20 ' enable full row select in list view control
Public Sub EnableListViewFullRowSelect(lvwLV As ListView)

    Dim lngStyle As Long
    Dim lngL As Long

    'get the current ListView style
25    lngStyle = SendMessageLong(lvwLV.hwnd, LVM_GETEXTENDEDLISTVIEWSTYLE, 0&,
    0&)

    'set the extended style bit
    lngStyle = lngStyle Or LVS_EX_FULLROWSELECT

    'set the new ListView style
30    lngL = SendMessageLong(lvwLV.hwnd, LVM_SETEXTENDEDLISTVIEWSTYLE, 0&,
    lngStyle)

End Sub

```

```

' Word.cls
VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
5  END
    Attribute VB_Name = "MSWord"
    Attribute VB_GlobalNameSpace = False
    Attribute VB_Creatable = True
    Attribute VB_PredeclaredId = False
10  Attribute VB_Exposed = False
    Option Explicit

    Private Const WM_CLOSE = &H10
    Private mWDApp As Word.Application

    Private Type RECT
15     left As Long
        top As Long
        right As Long
        bottom As Long
    End Type

    Private Declare Function GetParent Lib "user32" _
20     (ByVal hWndChild As Long) As Long

    Private Declare Function SetParent Lib "user32" _
        (ByVal hWndChild As Long, ByVal hWndNewParent As Long) As Long

    Private Declare Function FindWindow Lib "user32" _
25     Alias "FindWindowA" (ByVal lpClassName As String, _
        ByVal lpWindowName As String) As Long

    Private Declare Function SendMessage Lib "user32" _
        Alias "SendMessageA" _
        (ByVal hwnd As Long, ByVal wParam As Long, _
30     ByVal lParam As Long, lParam As Any) As Long

    Private Declare Function GetWindowRect Lib "user32" _
        (ByVal hwnd As Long, lpRect As RECT) As Long

    Private Declare Function SetWindowPos Lib "user32" _
        (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
35     ByVal X As Long, ByVal Y As Long, ByVal cx As Long, _
        ByVal cy As Long, ByVal wFlags As Long) As Long

```

```

Dim mlngHandle As Long
Dim origParent As Long
Dim origLeft As Long
Dim origTop As Long
5 Dim origWidth As Long
Dim origHeight As Long

```

```
Private Sub Class_Initialize()
```

```
    ' mlngHandle = FindWindow("OpusApp", vbNullString)
```

```
    ' Do While mlngHandle <> 0
```

```
10    '     SendMessage mlngHandle, WM_CLOSE, mlngHandle, 0
```

```
    '     mlngHandle = FindWindow("OpusApp", vbNullString)
```

```
    ' Loop
```

```
    mlngHandle = FindWindow("OpusApp", vbNullString)
```

```
    If mlngHandle <> 0 Then
```

```
15        Set mWdApp = GetObject(, "Word.Application.8")
```

```
    Else
```

```
        On Error Resume Next
```

```
        Set mWdApp = GetObject(, "Word.Application.8")
```

```
        If err.Number = 0 Then
```

```
20            MsgBox "Phantom WinWord detected!"
```

```
            Call mWdApp.Quit(False)
```

```
        Else
```

```
            err.Clear
```

```
        End If
```

```
25        Set mWdApp = CreateObject("Word.Application.8")
```

```
    End If
```

```
    mlngHandle = FindWindow("OpusApp", vbNullString)
```

```
    If mlngHandle <> 0 Then
```

```
        origParent = GetParent(mlngHandle)
```

```
30    If mWdApp.left < 0 Then
```

```
        origLeft = 0
```

```
    Else
```

```
        origLeft = mWdApp.left
```

```
    End If
```



```
    If mWdApp.top < 0 Then
        origTop = 0
    Else
        origTop = mWdApp.top
    End If
```

```
    origWidth = mWdApp.Width
    origHeight = mWdApp.Height
```

```
    Call SetParent(mlngHandle, frmTCA.fraWord.hwnd)
End If
```

```
10    mWdApp.Visible = True
```

```
End Sub
```

```
Private Sub Class_Terminate()
```

```
    mWdApp.Visible = False
```

```
    Call SetParent(mlngHandle, origParent)
    Call mWdApp.Move(origLeft, origTop)
    Call mWdApp.Resize(origWidth, origHeight)
```

```
    Call mWdApp.Quit(False) ' don't save!
```

```
End Sub
```

```
Public Property Get WordApp() As Word.Application
```

```
20    Set WordApp = mWdApp
```

```
End Property
```

```
Public Property Get DocumentsCount() As Long
```

```
    DocumentsCount = mWdApp.Documents.Count
```

```
End Property
```

```
25 Public Property Get SelectionType() As Long
```

```
    SelectionType = mWdApp.Selection.Type
```

```
End Property
```

```
Public Property Get SelectionText() As String
```

```
    SelectionText = mWDApp.Selection.Text
```

```
End Property
```

```
Public Sub Resize()
```

```
5    Dim WindowRect As RECT
```

```
    GetWindowRect frmTCA.fraWord.hwnd, WindowRect
```

```
    Dim lngH As Long
```

```
    Dim lngW As Long
```

```
10    lngW = frmTCA.ScaleX(WindowRect.right - WindowRect.left, vbPixels, vbPoints)  
    lngH = frmTCA.ScaleY(WindowRect.bottom - WindowRect.top, vbPixels, vbPoints)
```

```
    Call mWDApp.Resize(lngW, lngH)
```

```
    Call mWDApp.Move(0, 0)
```

```
15    ' SetWindowPos mlngHandle, 0, 0, 0, _  
    '     WindowRect.right - WindowRect.left, _  
    '     WindowRect.bottom - WindowRect.top, 64
```

```
    CommandBars("File").Controls("Exit").Enabled = False
```

```
End Sub
```

```
Public Sub CloseAllDocs()
```

```
    Dim docD As Document
```

```
20    For Each docD In mWDApp.Documents
```

```
        If Not docD.ReadOnly Then
```

```
            docD.Close
```

```
        Else
```

```
            Call docD.Close(False)
```

```
25    End If
```

```
    Next docD
```

```
End Sub
```

```
→
```

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT EXAMINING OPERATION

ATTN'Y DOCKET NO.: ETS-TCA

APPLICATION OF: PETER BRITTINGHAM, MARY E. MORLEY, MARK K.
SINGLEY, MARK G. ZELMAN, KRISHNA N. JHA,
JAMES H. FIFE, ROBERT L. RARICH, IRVIN R.
KATZ, RANDY E. BENNETT

FOR: COMPUTER-BASED TEST-ITEM GENERATION AND
CLONING

PROLOG SOURCE CODE APPENDIX
(PROLOG SCA 1-95)

PROLOG SOURCE CODE APPENDIX
TABLE OF CONTENTS¹

HLP4lib.p4	PROLOG SCA -1-
PrlgExpr.l	PROLOG SCA -13-
5 PrlgExpr.y	PROLOG SCA -16-
hlp4API.h	PROLOG SCA -90-

¹ All software COPYRIGHT 1999 ETS

' HLP4lib.p4

%%

%% HLP4lib.p4: library of PrologIV accessory relations useful in High-Level API

%%

5 %% We follow the convention that the [conventional] result is bound to the first argument,
%% so that these relations can be easily called as functions.

%%

%% Note: BEWARE: Interval operators (e.g. ./., .*., ...) sometimes give rise to errors (e.g.
divide-by-zero error)

10 %% in VB when used in conjunction with VB.

%% Note: BEWARE: intsplit/realsplit on unbounded variables sometimes give rise to errors
(e.g. overflow error)

%% in VB when used in conjunction with VB.

%% HLAPI-library functions/relations

15 %% debug_print(-Result, +List): Result is true if it prints all the elements of
the list, ended by nl

debug_print(true, []) :- nl.

debug_print(Result, [A| Rest]) :-

write(A), write(','),

20 debug_print(Result, Rest).

%% var_with_precision(+V, +Prec): Succeeds if var V is an integer multiple of
given Precision.

%% (Do NOT introduce intsplit(N) here - gives rise to overflow error in VB.)

var_with_precision(V, Prec) :-

25 int(N),

V = N*Prec.

%% even(-Result, +N): Result is true if N is an even integer.

even(true, N) :-

30 even(N).

%% odd(-Result, +N): Result is true if N is an odd integer.

odd(true, N) :-

odd(N).

35 %% max(-Result, +A, +B): Result is maximum of A and B. -- already provided in
PrologIV

%% max(-Result, [A, B, C, ...]): Result is maximum of array of numbers [A, B, C,
...].

max(Result, [A| Rest]) :-

max_array_aux(Result, A, Rest).

40 max_array_aux(A, A, []).

```

max_array_aux(Result, A, [B|Rest]) :-
    max(Rmax, A, B),
    max_array_aux(Result, Rmax, Rest).

%% min(-Result, +A, +B): Result is minimum of A and B. -- already provided in
5 PrologIV
%% min(-Result, [A, B, C, ...]): Result is minimum of array of numbers [A, B, C,
...].
min(Result, [A|Rest]) :-
    min_array_aux(Result, A, Rest).
10 min_array_aux(A, A, []).
min_array_aux(Result, A, [B|Rest]) :-
    min(Rmin, A, B),
    min_array_aux(Result, Rmin, Rest).

%% mean(-Mean, A, B): Mean is mean of numbers A & B
15 mean((A+B)/2, A, B).

%% mean(-Mean, [A, B, C, ...]): Mean is mean of array of numbers [A,B,C,...]
mean(A, [A]).
mean(Sum/Size, [A|Rest]) :-
    array_sum(Sum, [A|Rest]),
20 size(Size, [A|Rest]).

%% median(-Med, +[A, B, C, ...]): Med is the median of array of numbers [A, B,
C, ...]
median(Med, [A|Rest]) :-
    sort(SortedList, [A|Rest]),
    size(Size, SortedList),
    pick_midlist(Med, SortedList, Size).
pick_midlist(Mid, List, ListSize) :-
    odd(ListSize),
    index(Mid, List, ((ListSize-1)/2)+ 1).
30 pick_midlist((Mid1+Mid2)/2, List, ListSize) :-
    even(ListSize),
    index(Mid1, List, ListSize/2),
    index(Mid2, List, (ListSize/2)+1).

%% gcd(-GCD, +A, +B): GCD is gcd of A and B. -- until PrologIV provides
35 it.
gcdtemp(A./Num, A, B) :-
    int(A), int(B), B gtlin 0,
    numden(A./B, Num, _Den).

%% lcm(-LCM, +A, +B): LCM is lcm of A and B. (lcm(A, B)= A* B/

```

```

gcd(A,B) )-- until PrologIV provides it.
lcmtemp((A.*B)/.GCD, A, B)      :-
    int(A), int(B), B gtlin 0,
    gcdtemp(GCD, A, B).

```

```

5      %% mod(-Mod, N, D): Mod= N modulo M.
modtemp(Mod, N, D)              :-
    int(N), int(D), D gtlin 0,
    modulo(N, D, Mod).

```

```

10     %% divmod(-DivMod, N, D): True if DivMod= [N Div D, N Mod D]
divmod([Div, Mod], N, D)      :-
    int(N), int(D), D gtlin 0,
    intdiv(Div, N, D),
    modulo(N, D, Mod).

```

```

15     %% numdentemp([-N, -D], R): True if N/D = R (where N and D are integers)
numdentemp([N, D], R)        :-
    real(R),
    numden(R, N, D).

```

```

20     %% quotnumden([-Q, -N, -D], R): True if (Q+(N/D)) = R (where Q, N and D are
integers)
quotnumden([Q, N, D], R)     :-
    real(R),
    numden(R, N1, D),
    divmod([Q, N], N1, D).

```

```

25     %% sqrt(Sqrt, N):      Sqrt is square-root of N -- already provided by PrologIV
    %% is_perfect_square(-Result, +N): succeeds (and sets Result to true) if N is a
perfect square.

```

```

is_perfect_square(true, N)    :-
    int(N), int(Sqrt),
    sqrt(Sqrt, N).

```

```

30     %% isnot_perfect_square(-Result, +N): succeeds (and sets Result to true) if N is
NOT a perfect square.

```

```

isnot_perfect_square(true, N) :-
    int(N), nint(Sqrt),
    sqrt(Sqrt, N).

```

```

35     %% cubert(-CubeRoot, +N): CubeRoot is cube-root of N
cubert(CubeRt, N)            :-
    int(N),
    root(CubeRt, N, 3).

```

```

    %% is_perfect_cube(-Result, +N): succeeds (and sets Result to true) if N is a

```

perfect cube.

is_perfect_cube(true, N) :-

int(N), int(Cbrt),

cubert(Cbrt, N).

5 %% isnot_perfect_cube(-Result, +N): succeeds (and sets Result to true) if N is
NOT a perfect cube.

isnot_perfect_cube(true, N) :-

is_perfect_cube(true, N), !, fail.

10 isnot_perfect_cube(true, N) :- %% this alone does not work well because of roundoff
problems

int(N), nint(Cbrt),

cubert(Cbrt, N).

 %% is_prime(-Result, +N): succeeds (and sets Result to true) if N is a
prime-number.

15 is_prime(true, 2). is_prime(true, 3). %% base cases (note that 1 is NOT considered prime.)

is_prime(true, N) :-

int(N), abs(AbsN, N), AbsN gt 3,

sqrt(RlSqRoot, AbsN),

ceil(SqRoot, RlSqRoot),

20 aux_check_prime(AbsN, 2, SqRoot).

aux_check_prime(N, CurDivisor, MaxDivisor) :-

CurDivisor gt MaxDivisor.

aux_check_prime(N, CurDivisor, MaxDivisor) :- %% improve it later

CurDivisor le MaxDivisor,

modulo(N, CurDivisor, Mod), Mod gt 0,

25 aux_check_prime(N, CurDivisor + 1, MaxDivisor).

 %% isnot_prime(-Result, +N): succeeds (and sets Result to true) if N is NOT a
prime-number.

30 isnot_prime(true, N) :-

nint(N), real(N).

isnot_prime(true, N) :-

int(N), abs(AbsN, N), AbsN gt 3,

sqrt(RlSqRoot, AbsN),

ceil(SqRoot, RlSqRoot),

35 aux_check_nonprime(AbsN, 2, SqRoot).

aux_check_nonprime(N, CurDivisor, MaxDivisor) :-

CurDivisor le MaxDivisor,

modulo(N, CurDivisor, 0), !.

40 aux_check_nonprime(N, CurDivisor, MaxDivisor) :-

CurDivisor le MaxDivisor,

aux_check_nonprime(N, CurDivisor + 1, MaxDivisor).

 %% nth(-NthElem, +N, +List): NthElem is the Nth element of the given List;

(N==1 for first elem.)

```
nth(NthElem, N, List) :-  
    index(NthElem, List, N).
```

```
5      %% permute(-PermutedList, +List): PermutedList is a permutation of List.  
permute(PermutedList, List) :-  
    permute_aux(PermutedList, [], List).
```

```
permute_aux(PermutedList, PermutedList, []).  
permute_aux(PermutedList, APermutation, RightList) :-  
10    an_elem_and_rest(Elem, Rest, RightList),  
    permute_aux(PermutedList, [Elem| APermutation], Rest).
```

```
    %% factorial(-Factorial, +N): Factorial= N!  
    %% naive implementation: factorial(1, 0). factorial(N* Fact, N) :-  
15    factorial(Fact, N-1).
```

```
    %% we just list them here which also gives a bidirectional relationship.  
factorial(1, 0). factorial(1, 1). factorial(2, 2).  
factorial(6, 3). factorial(24, 4). factorial(120, 5).  
factorial(720, 6). factorial(5040, 7). factorial(40320, 8).  
20 factorial(362880, 9). factorial(3628800, 10).  
factorial(39916800, 11). factorial(479001600, 12).  
factorial(6227020800, 13). factorial(87178291200, 14).  
factorial(1307674368000, 15). factorial(20922789888000, 16).  
factorial(355687428096000, 17). factorial(6402373705728000, 18).  
25 factorial(121645100408832000, 19). factorial(2432902008176640000, 20).  
factorial(N* Fact, N) :- number(N), N gt 20, factorial(Fact, N-1).
```

```
    %% non-naive implementation of factorial - not used  
%% factorial(1, 0).  
%% factorial(Factorial, N) :-  
30 %% int(N), N gt 0,  
%% factorial(Factorial, N, N.-.1).  
%% factorial(Factorial, Factorial, 0).  
%% factorial(Factorial, FactSoFar, N) :-  
%% N gt 0,  
35 %% factorial(Factorial, FactSoFar*.N, N.-.1).
```

```
    %% enumerate(-R, +Min, +Max, +Step): enumerate (any var) R between  
(closed-interval) [Min, Max] by Step.  
enumerate(R, Min, Max, Step) :-  
40    min(RMin, Min, Max),  
    max(RMax, Min, Max),  
    RMin le RMax,
```

```

Step gt 0,
enumerate_aux(R, RMin, RMax, Step).

```

```

%% enumerate_int(-I, +Min, +Max, +Step): enumerate (integer var) I between
(closed-interval) [Min, Max] by Step.

```

```

5 enumerate_int(I, Min, Max, Step) :-
    int(I),
    min(RMin, Min, Max),
    max(RMax, Min, Max),
    RMin le RMax,
10    ceil(IMin, RMin),
    floor(IMax, RMax),
    IMin le IMax,
    floor(ISTep, Step),    %% should IStep be floor, or ceiling ??
    IStep gt 0,
15    enumerate_aux(I, IMin, IMax, IStep).

```

```

%% enumerate_aux(-R, +Min, +Max, +Step): enumerate (any var) R between
(closed-interval) [Min, Max] by Step.

```

```

%% Note that we enumerate from both ends i.e. from Min and from Max ends.

```

```

%% Note that increasing the no. of partitions requires large choice-stack and heap
sizes.

```

```

%% (We can adjust the various stack & heap sizes - but just that there is a cost to more
partitions.)

```

```

enumerate_aux(R, Min, Max, Step) :-
    StepsCnt= (Max- Min)/Step,
25    StepsCnt le 4,    %% le 4 partitions => le 5 points in the range
    !,                %% small range - simple enumeration
    enumerate_aux_simple(R, Min, Min, Max, Step).
enumerate_aux(R, Min, Max, Step) :-    %% large range - interleave the enumeration
    StepsCnt= (Max- Min)/Step,    %% adjust the Max
30    floor(NMax, StepsCnt),
    PartitionStepCnt = StepsCnt/6,    %% split the steps-range into 5 partitions
    floor(Inc, PartitionStepCnt),    %% problem with ceil/2 when numbers are
small

```

```

    enumerate_aux_interleave(R,    %% interleave the 10 partitions
35    [[Min, Min+ Inc* Step, inc],    %% 1st: enumerate up
    [Min+(Inc+1)*Step, Min+2*Inc*Step, dec], %% 2nd: enumerate down
    [Min+(2*Inc+1)*Step, Min+3*Inc*Step, inc],
    [Min+(3*Inc+1)*Step, Min+4*Inc*Step, dec],
    [Min+(4*Inc+1)*Step, Min+NMax*Step, dec]],
40    Step).

```

```

%% enumerate R simply between Min & Max by Step
enumerate_aux_simple(R, R, _Min, _Max, _Step).

```

```

enumerate_aux_simple(R, Prev, Min, Max, Step) :-
    (Prev+ Step) le Max,
    enumerate_aux_simple(R, Prev+Step, Min, Max, Step).

    %% enumerate in an interleaved fashion - from all partitions - upward or
5 downward
enumerate_aux_interleave(RMin, [[RMin, RMax, inc]] _Rest], _Step):-
    RMin le RMax.
enumerate_aux_interleave(RMax, [[RMin, RMax, dec]] _Rest], _Step):-
    RMin le RMax.

10 enumerate_aux_interleave(R, [[RMin, RMax, _IncOrDec]] Rest], Step):-
    (RMin+ Step) gt RMax, !,    %% partition done - drop it from the partitions-list
    random_shuffle(Shuffled, Rest),
    enumerate_aux_interleave(R, Shuffled, Step).

15 enumerate_aux_interleave(R, [[RMin, RMax, inc]] Rest], Step):-
    random_shuffle(Shuffled, Rest),
    ncons(ResLst, [RMin+ Step, RMax, inc], Shuffled),
    enumerate_aux_interleave(R, ResLst, Step).
enumerate_aux_interleave(R, [[RMin, RMax, dec]] Rest], Step):-
20 random_shuffle(Shuffled, Rest),
    ncons(ResLst, [RMin, RMax- Step, dec], Shuffled),
    enumerate_aux_interleave(R, ResLst, Step).

    %% select_r_of_n_ordered(-Pn_r, +N, +R): Select no. of permutations of setsize
25 N selecting R at a time
select_r_of_n_ordered(Nfact./ Rfact, N, R) :-
    int(N), int(R), N gt 0, R gt 0, N ge R,
    factorial(Nfact, N),
    factorial(Rfact, R).

30    %% select_r_of_n(-Pn_r, +N, +R): Select no. of combinations of setsize N
    selecting R at a time
select_r_of_n(Perm./ NRfact, N, R) :-
    select_r_of_n_ordered(Perm, N, R),
35 factorial(NRfact, N-R).

    %% abs(-AbsN, +N): AbsN is absolute number, N. -- already provided by
PrologIV

    %% isbound(-Result, +X): Result is true if the given (numerical variable/value) X
40 is bound on the low and/or upper side.

```

```

isbound(true, X)      :-      isbound(X).
isbound(X) :- glb(X,_) , !.
isbound(X) :- lub(X,_) .

```

```

%% isnotbound(-Result, +X): Result is true if the given (numerical variable) X is
NOT bound on either the low or the upper side.

```

```

isnotbound(true, X)      :-      isnotbound(X).
isnotbound(X) :- isbound(X), !, fail.
isnotbound(X).

```

```

%% auxiliary functions

```

```

%% even(N): true if N is even
even(2.*.X)      :-      int(X).

```

```

%% odd(N): true if N is odd
odd(N)           :-      int(N), N= 2.*.X, nint(X).

```

```

%% reverse(-ReverseList, +List): ReverseList is reverse-ordered List
reverse(ReverseList, List)      :-
    reverse(ReverseList, [], List).
reverse(ReverseList, ReverseList, []).
reverse(ReverseList, RevListSoFar, [A|Rest])      :-
    reverse(ReverseList, [A|RevListSoFar], Rest).

```

```

%% ncons(-ResList, +A, +List): true if ResList= List+ [A].
ncons(ResList, A, List)      :-
    append(ResList, List, [A]).

```

```

%% append(-AppendedList, List1, List2): AppendedList is result of appending
lists List1 & List2.
append(A, [], A).
append([A|Result], [A|Rest], B)      :-
    append(Result, Rest, B).

```

```

%% random_shuffle(-ShuffledList, +List): Succeeds if List is random-shuffled
into SuffledList
random_shuffle(ShuffledList, List)      :-
    random_shuffle(ShuffledList, [], List).

```

```

random_shuffle(ShuffledList, ShuffledList, []).
random_shuffle(ShuffledList, ShuffledListSoFar, [A| Rest])      :-
    brandom(Random),
    ((Random=1) ->
    random_shuffle(ShuffledList, [A|ShuffledListSoFar], Rest);
    (ncons(NewShuffledListSoFar, A, ShuffledListSoFar),

```

```

random_shuffle(ShuffledList, NewShuffledListSoFar, Rest) )
).

```

```

5      %% array_sum(-ArraySum, +Array): Sum is the sum of the array-elements of
      Array
      array_sum(0, []).
      array_sum(ArraySum, [A|Rest]) :-
          number(A),
10      array_sum(ArraySum, A, Rest).
      array_sum(ArraySum, ArraySum, []).
      array_sum(ArraySum, SumSoFar, [A|Rest]) :-
          number(A),
          array_sum(ArraySum, SumSoFar+.A, Rest).

15      %% sort(-SortedArray, +Array): Array (of numbers) is sorted (in ascending order)
      into SortedArray
      sort([], []).
      sort(SortedArray, [E|Array]) :-                %% quicksort
          partition_mine(Smaller, Greater, E, Array),
20      sort(SortedSmaller, Smaller),
          sort(SortedGreater, Greater),
          append(SortedArray, SortedSmaller, [E|SortedGreater]).

      %% partition(-Smaller, -Greater, +Elem, +Array): Partition the Array (of
      numbers) into
25      %%      subarrys Smaller and Greater than Elem.
      partition_mine([], [], _, []).
      partition_mine([Small|Smaller], Greater, Elem, [Small|Array]) :-
          Small le Elem,                %% Small <= Elem,
          partition_mine(Smaller, Greater, Elem, Array).
30      partition_mine(Smaller, [Great|Greater], Elem, [Great|Array]) :-
          Great gt Elem,                %% Great > Elem
          partition_mine(Smaller, Greater, Elem, Array).

      %% rotate(-RotatedList, +List, +N): Rotate the given List by N steps into
      RotatedList.
35      rotate([], [], _).
      rotate(RotatedList, List, N) :-
          first_N_elems(FirstNElems, RestElems, List, N),
          append(RotatedList, RestElems, FirstNElems).

40      %% first_N_elems(-FirstNElems, -RestElems, +List, +N): true if List =
      FirstNElems + RestElems.
      first_N_elems([], List, List, 0).

```

```

first_N_elems([], [], [], _N).
first_N_elems([A| NextElems], RestElems, [A| Rest], N)   :-
    int(N), N gt 0,
    first_N_elems(NextElems, RestElems, Rest, N.-.1).

```

5 %% one_of(Elem, List): true if Elem is an element of the given List

```

%%one_of(Elem, List)          :-      %% this implementation has some disadvantages
%%      inlist(Elem, List).
one_of(Elem, [Elem| _]).
one_of(Elem, [_|List])        :-
10      one_of(Elem, List).

```

```

%% not_one_of(Elem, List): true if Elem is not an element of the given List
not_one_of(Elem, List)        :-
    outlist(Elem, List).

```

15 %% an_elem_and_rest(Elem, Rest, List): true if Elem is an element of the List,
and Rest= List- Elem.

```

an_elem_and_rest(Elem, Rest, List) :-
    an_elem_and_rest_aux(Elem, Rest, [], List).
an_elem_and_rest_aux(A, Rest, LeftList, [A| Remainder]) :-
20      append(Rest, LeftList, Remainder).
an_elem_and_rest_aux(Elem, Rest, LeftList, [A|Remainder])    :-
    append(NewLeftList, LeftList, [A]),
    an_elem_and_rest_aux(Elem, Rest, NewLeftList, Remainder).

```

25 %% random(-Elem, +List): return a random element (Elem) from the given list

```

random(Elem, List)  :-
    list(List),
    random_shuffle(ShuffledList, List),
30      one_of(Elem, ShuffledList).

```

35 %% random(-Random, +Range): return a random number in the range 0 ... Range

```

random(Rand, Range)  :-
    int(Range), Range> 0,
    random(R),
    modulo(R, Range+1, Rand).

```

40 %% brandom(-BRandom): True if BRandom is a binary (i.e. 0 or 1) random number

```

brandom(BRand)      :-
    random(Rand),
    (((Rand/2147483647)>= 0.5) -> BRand = 1; BRand = 0).

```

```

        %% random(-Random): True if Random is a (pseudo-) random integer
        %%
        %% Note that this works because PrologIV can handle numbers greater than 32-bits
randomize(S) :- integer(S), record(seed, S).
5 random(X) :-
    recorded(seed, S),
    (integer(S) -> S1 = S; S1 = 1),
    Z = S1*16807,
    modulo(Z, 2147483647, X),
10    record(seed, X).

%% KNJ: Seems like a very inadequate randomizer: we get numbers 7n+4 from it !!
%%      As such, you will have (for this randomizer): random()% 7 = 4.
%% --- Not used any more : KNJ ---
%%      KNJ: Modified the multiplier to a prime-no, and that seems to have improved the
15 generator.
%%%%%%%%%%
% Random number generator from Pascal Bouvier of PrologIA :
%-- random.p4 --
% un generateur de nombres pseudo-aleatoires.
20 % A pseudo-random number generator.
% (formula got from C-ANSI random() ?)
%
% Pascal Bouvier, d'apres Prolog III
% (c) PrologIA 1996,1997

25 % initializer
orandomize(S) :- integer(S) ,!, record(oseed, S).
orandomize(S) :- var(S), record(oseed, 1).
orandom(X) :-
30    recorded(oseed,S),
    (integer(S) -> S1 = S ; S1 = 1),
    Z = S1*1103515245 + 12345,
    modulo(Z, 65536*65536-1, Z1),
    record(oseed,Z1),
    modulo(Z,32767, X).

35 %%%%%%%%%%%

```

PROLOG SCA -12-


```

' PrgExpr.l
/*
 * PrgExpr.l: Lexical analyzer for constraints:
 *           (splits constraints into lexical-components e.g. words, punctuations).
5  *
 */
%{
#include <math.h>
#include <malloc.h>
10 #include <string.h>
#include "p4term.h"
#include "prlgHLAPI.h"
#include "PrgExpr.tab.h"

extern int GetInString(char *buf, int max_size);
15 //extern int yylval;    -- supplanted by "extern YYSTYPE yylval;" in PrgExpr.tab.h

#define YY_INPUT(buf, result_cnt, max_size)    {if (!(result_cnt= GetInString(buf,
max_size))) {buf[0]= YY_NULL; result_cnt= 1;} }

#define MAX_VAL_BUF                64
static char valBuf[64][MAX_VAL_BUF];
20 static int valBuf_x= 0;
#define New_VAL_BUF                ((valBuf_x< MAX_VAL_BUF)? valBuf[valBuf_x++]:
(valBuf_x= 0, valBuf[valBuf_x++]))
//define STRDUP(str)                (strcpy(New_VAL_BUF, (str)))
#define STRDUP(str)                (strdup(str))

25 %}

/*
Note that, in flex, predefined char. classes (which must appear within [ ]) include:
UPPER      [A-Z]      [:upper:]
30 LOWER    [a-z]      [:lower:]
ALPHA      [a-zA-Z]    [:alpha:]
ALNUM      [A-Za-z0-9] [:alnum:]
DIGIT      [0-9]      [:digit:]
SPACE      [\n\r\t\v\f\O ] [:space:]
*/

35 %%
"end_var_defs"                {yylval.ival= END_VAR_DEFS;
return(END_VAR_DEFS);}
"freeze"                      {yylval.ival= FREEZE;
return(FREEZE);}

```

	"succeed"	{yyval.ival= SUCCEED;}
	return(SUCCEED);}	
	"fail"	{yyval.ival= FAIL;}
	return(FAIL);}	
5	"if"	{yyval.ival= IF;}
	return(IF);}	
	"then"	{yyval.ival= THEN;}
	return(THEN);}	
	"else"	{yyval.ival= ELSE;}
10	return(ELSE);}	
	"elseif"	{yyval.ival= ELSEIF;}
	return(ELSEIF);}	
	"int("	{yyval.ival= INT_PRED;}
	return(INT_PRED);}	
15	"real("	{yyval.ival= REAL_PRED;}
	return(REAL_PRED);}	
	"fraction("	{yyval.ival= FRACTION_PRED;}
	return(FRACTION_PRED);}	
	"list("	{yyval.ival= LIST_PRED;}
20	return(LIST_PRED);}	
	"eq_vars("	{yyval.ival= EQVARS_PRED;}
	return(EQVARS_PRED);}	
	"neq_vars("	{yyval.ival= NEQVARS_PRED;}
	return(NEQVARS_PRED);}	
25	"neq_varvals(" {yyval.ival= NEQVARVALS_PRED;}	
	return(NEQVARVALS_PRED);}	
	"optimizable_rel(" {yyval.ival= OPTIMIZABLEREL_PRED;}	
	return(OPTIMIZABLEREL_PRED);}	
	"step"	{yyval.ival= STEP;}
30	return(STEP);}	
	"symbol(" {yyval.ival=	
	SYMBOL_PRED; return(SYMBOL_PRED);}	
	"pi"	{yyval.fval=
	(float)PI; return(PI);}	
35	"in"	{yyval.ival=
	IN_SET; return(IN_SET);}	
	"from"	{yyval.ival=
	FROM_SET; return(FROM_SET);}	
	"notin"	{yyval.ival=
40	NOTIN_SET; return(NOTIN_SET);}	
	"not"	{yyval.ival= NOT;}
	return(NOT);}	
	"=="	{yyval.ival= EQ;}
	return(EQ);}	
45	"!="	{yyval.ival= NEQ;}

```

return(NEQ);}
">="                                {yyval.ival= GE;
return(GE);}
"<="                                {yyval.ival= LE;
5  return(LE);}
"["                                  {yyval.ival=
EXRANGE_START; return(EXRANGE_START);}
[[:upper:]]_ [[:alnum:]]_*           {yyval.string= STRDUP(yytext);
return(VAR);}
10 [[:lower:]] [[:alnum:]]_*          {yyval.string= STRDUP(yytext);
return(ATOM_CONST);}
[[:digit:]]*" [[:digit:]]+           {yyval.string= STRDUP(yytext); /* stuffs
string-rep */ return(REALNUM);}
[[:digit:]]+                         {yyval.ival= atoi(yytext);
15 return(INTNUM);}
[[:space:]]+                         {continue;}
.                                    {yyval.ival=
yytext[0]; return(yytext[0]);}
%%
20 int yywrap() {return(1);}

```

```

' PrlgExpr.y
%{
/*
 * PrlgExpr.y: Parser for Prolog constraints - to provide functionality for
5  *      high-level communication, using mathematical expressions, between
 *      Prolog IV and other programs (e.g. TCA-GUI).
 *      (Use: bison -vd PrlgExpr.y to process it.)
 *
/

10  #include <stdio.h>
#include <time.h>
#include <string.h>
#include <malloc.h>
#include "p4term.h"
15  #include "prlgHLAPI.h"

#define CHECK_CNT                                // <<-- define
it to check any buffer-overflows
#define VERSION          "3.3f"                // <<-- update it appropriately
#define P4HLAPILIB      "HLP4lib.p4"

20  #define TRUE          1
#define FALSE           0
#define PI_VAL          (3.14159265)
#define MAX(a, b)      ((a)>=(b)? (a):(b))
#define MIN(a, b)      ((a)<=(b)? (a):(b))
25  #define ABS(x)        ((x)>= 0? (x): -(x))
#define list(elem)      (cons((elem), NULL))
#define SAME(x, y)      (ABS((x)-(y))<= (Precision))

// sizes for some of P4 stacks in terms of cells (1 cell = 8
bytes)
30  #define P4HEAP_SIZE          (1000000)          /*
P4-heap-stack-size in cell (= 8 bytes) counts [default: 700000] */
#define P4CHOICE_SIZE          (300000)          /*
P4-choice-stack-size in cell (= 8 bytes) counts [default: 50000] */

#define p4val_rational(T)      (p4val_as_double(T))
35  #define p4val_cstring(T)     (p4_symbol_to_cstring(p4val_symbol(T)))

#ifdef CHECK_CNT
static P4TERM check_term(P4TERM term)\
{if (!term) printf("\n***Received NULL P4TERM***\n");\
if (p4errno!= 0) {printf("\n***Unknown error occurred before the given term was

```

```

checked;***\n");fflush(stdout);}\
return term;}
#else
#define check_term(term)                (term)
5 #endif /* CHECK_CNT */
#define P4MAKE_FUNC_0(func_str)
    check_term(p4make_atom(p4str2symbol(func_str)))
#define P4MAKE_FUNC_1(func_str, arg)
    check_term(p4make_functor(1, p4str2symbol(func_str), (arg)))
10 #define P4MAKE_FUNC_2(func_str, arg1, arg2)    check_term(p4make_functor(2,
    p4str2symbol(func_str), (arg1), (arg2)))
#define P4MAKE_FUNC_3(func_str, arg1, arg2, arg3)
    check_term(p4make_functor(3, p4str2symbol(func_str), (arg1), (arg2), (arg3)))
#define P4MAKE_FUNC_4(func_str, arg1, arg2, arg3, arg4)
15 check_term(p4make_functor(4, p4str2symbol(func_str), (arg1), (arg2), (arg3), (arg4)))

#define P4AND(arg1, arg2)                ((arg1) &&
    (arg2)? P4MAKE_FUNC_2(",", arg1, arg2): (arg1)? (arg1): (arg2))
#define P4COMMA(arg1, arg2)            (P4AND(arg1,
    arg2))
20 #define P4OR(arg1, arg2)                ((arg1) &&
    (arg2)? P4MAKE_FUNC_2(";", arg1, arg2): (arg1)? (arg1): (arg2))
#define P4EQ(arg1, arg2)                ((arg1) &&
    (arg2)? P4MAKE_FUNC_2("=", arg1, arg2): (arg1)? (arg1): (arg2))
#define P4NEQ(arg1, arg2)                ((arg1) &&
25 (arg2)? P4MAKE_FUNC_2("dif", arg1, arg2): (arg1)? (arg1): (arg2))
#define P4IF_THEN(cond_t, then_t)        (P4MAKE_FUNC_2("->", cond_t,
    then_t))
#define P4IF_THEN_ELSE(cond_t, then_t, else_t)    (P4OR(P4IF_THEN(cond_t, then_t),
    else_t))
30 #define P4IF_THEN_ELSEIF(then_cond_t, then_t, else_cond_t, else_t)
    (P4OR(P4AND(then_cond_t, then_t), P4AND(else_cond_t, else_t)))
#define P4NOT(arg)
    (P4MAKE_FUNC_1("\\+", arg))
#define P4CUT
35 (P4MAKE_FUNC_0("!"))
#define P4TRUE
    (P4MAKE_FUNC_0("true"))
#define P4FAIL
    (P4MAKE_FUNC_0("fail"))
40 #define P4FALSE
    (P4FAIL)

    // need to wrap goal in p4call/1 before calling p4make_call() (to interpret +/2, -/2, ...)
#define P4MAKE_CALL(goal)    (p4make_call(P4MAKE_FUNC_1("p4call", goal)))

```

```

// due to a bug in p4what_is(), need to call dereference
#define P4WHAT_IS(term)      (p4what_is(dereference(term)))

// max. size (in bytes) of an expression passed to the API
#define MAX_EXPR_SIZE      4096
5 // max no. of variables (in one call to Prolog IV)
#define MAX_VAR_CNT      1024
// max. length of variable name
#define MAX_VAR_NAME_LEN  32
// max-size of the funcTermBuf[]
10 #define MAX_FUNC_TERM_BUF_CNT  128
// max-size of the anonVarBuf[]
#define MAX_ANON_VAR_CNT  128
// max-size of the constBuf[]
#define MAX_CONST_CNT      128
15 // max. arity of a functor
#define MAX_ARITY      10

#define random      rand // MS VC does not
have random() -- remove when we can have random()
#define srand      srand // MS VC does not
20 have srand()-- remove when we can have random()
//extern void srand(long);
//extern long random();
extern double atof();
//extern int isspace();
25 extern int p4errno;

#ifdef PRLGHLAPI
__declspec(dllexport) long AbortPrologSoln; // Flag; if TRUE, Prolog
constraint-solving-process is aborted
#else
30 long AbortPrologSoln; // Flag; if TRUE, Prolog constraint-solving-process is aborted
#endif // PRLGHLAPI

//typedef char BOOLEAN;

static struct s_rename_struct // used in make_functor()
{
35 char *func_name, *map_to_name; // for mapping func-names e.g. gcd ->
gcdtemp
} FuncRenameList[] = {
    {"ceiling", "ceil"},
    {"gcd", "gcdtemp"},
40 {"lcm", "lcmtmp"},

```

```

        {"mod", "modtemp"},
        {"numden", "numdentemp"},
        {NULL, NULL}};    // make sure to end the list with {NULL, NULL}

typedef struct s_tabelem
5 { // an element of the name-key table
    char type; // user-specified type (e.g. int(X)) of the variable; 0 if any type will do.
        char ongrid; // flag: true if var-value is to be an integral multiple of its precision; false
otherwise
    double precision; // precision for the variable
10    char name[MAX_VAR_NAME_LEN+1]; // variable name
    P4TERM term; // P4 Prolog term representation for the variable
    struct s_tabelem *next;
        char is_independent_var; // TRUE if the variable is independent (& is not a
constant); FALSE otherwise
15 } TabElem;

typedef struct s_constant
{
    P4TERM term;
    Value value;
20 } Const;

static char inExprBuf[MAX_EXPR_SIZE+1]= {0}; // buffer to store the incoming
expression (string) in - for parsing purposes
static int inExprBuf_x= 0, inExprBuf_cnt= 0;
#define INIT_inExprBuf {inExprBuf_cnt= inExprBuf_x= 0;}

25 // buffer to store func-terms for arity conversion (i.e. X= func(Y, Z). -> X= _R,
func(_R, Y, Z).)
static P4TERM funcTermBuf[MAX_FUNC_TERM_BUF_CNT]= {NULL};
static int funcTermBuf_x= 0;
#define INIT_funcTermBuf {funcTermBuf_x= 0; memset(funcTermBuf, 0,
30 MAX_FUNC_TERM_BUF_CNT* sizeof(P4TERM));}
#ifdef CHECK_CNT
#define ADD_funcTermBuf(term) {if (funcTermBuf_x< MAX_FUNC_TERM_BUF_CNT)
funcTermBuf[funcTermBuf_x++]= (term); else {printf("\n***FUNC_TERM_BUF
overflow***\n"); fflush(stdout);}}
35 #else
#define ADD_funcTermBuf(term) {funcTermBuf[funcTermBuf_x++]= (term);}
#endif /* CHECK_CNT */
#define FOR_EACH_FUNC_TERM(term) {int _i_func; for(_i_func= 0; _i_func<
funcTermBuf_x && ((term)= funcTermBuf[_i_func]); _i_func++) {
40 #define END_FOR_EACH_FUNC_TERM(term) }}

```

```

// buffer for anonymous var's used in the functions
static P4TERM anonVarBuf[MAX_ANON_VAR_CNT]= {NULL};
static int anonVarBuf_x= 0;
#define INIT_anonVarBuf {anonVarBuf_x= 0; memset(anonVarBuf, 0,
5 MAX_ANON_VAR_CNT* sizeof(P4TERM));}
#ifdef CHECK_CNT
#define ADD_anonVarBuf(anon_term) {if (anonVarBuf_x< MAX_ANON_VAR_CNT)
anonVarBuf[anonVarBuf_x++]= (anon_term); else {printf("\n***ANON_VAR_BUF
10 overflow***\n");fflush(stdout);}}
#else
#define ADD_anonVarBuf(anon_term) {anonVarBuf[anonVarBuf_x++]= (anon_term);}
#endif /* CHECK_CNT */
#define FOR_EACH_ANON_VAR(anon_term) {int _i_anon; for(_i_anon= 0; (_i_anon<
anonVarBuf_x) && ((anon_term)= anonVarBuf[_i_anon]); _i_anon++) {
15 #define END_FOR_EACH_ANON_VAR(anon_term) }}

// buffer to store number-constant's (with their values)
static Const constBuf[MAX_CONST_CNT]= {0};
static int constBuf_x= 0;
#define INIT_constBuf {constBuf_x= 0;}
20 #ifdef CHECK_CNT
#define ADD_constBuf(trm, const_val) {if (constBuf_x< MAX_CONST_CNT)
{constBuf[constBuf_x].term= (trm); constBuf[constBuf_x].value= (const_val);constBuf_x++;}
else {printf("\n***CONST_BUF overflow***\n");fflush(stdout);}}
#else
25 #define ADD_constBuf(trm, const_val) {constBuf[constBuf_x].term= (trm);
constBuf[constBuf_x].value= (const_val); constBuf_x++;}
#endif /* CHECK_CNT */

// setup to allocate TabElem's from a circular buffer - quick, easy to reinitialize,
& no need to free up pointers
30 static TabElem tabSpace[MAX_VAR_CNT]= {0}; // space to alloc TabElem from
static int tabSpace_x= 0;
// alloc a new TabElem from a circular buffer
#ifdef CHECK_CNT
#define NEW_TAB_ELEM ((tabSpace_x< MAX_VAR_CNT)? &tabSpace[tabSpace_x++]:
35 (tabSpace_x= 0, printf("\n***tabSpace-buffer overflow***\n"), fflush(stdout),
&tabSpace[tabSpace_x++]))
#else
#define NEW_TAB_ELEM ((tabSpace_x< MAX_VAR_CNT)? &tabSpace[tabSpace_x++]:
40 (tabSpace_x= 0, &tabSpace[tabSpace_x++]))
#endif /* CHECK_CNT */
// [re]intialize the table
#define INIT_tabSpace {tabSpace_x= 0; memset(tabSpace, 0,
MAX_VAR_CNT*sizeof(TabElem));}

```



```

// [hash] table for Var's
#define VAR_TABLE_SIZE 53
static TabElem *varTable[VAR_TABLE_SIZE] = {NULL};

// varList: array of var's from the current constraint e.g. X, Y from "X= Y+2,
Y=6."
static TabElem *varList[MAX_VAR_CNT]= {NULL};
static int varList_x= 0;
// add the given TabElem-ptr to the varList
#ifdef CHECK_CNT
#define ADD_varList(p_tabElem)  {if (varList_x< MAX_VAR_CNT) varList[varList_x++]=
10 (p_tabElem); else {printf("\n***varList-buffer overflow***\n");fflush(stdout);}}
#else
#define ADD_varList(p_tabElem)  {varList[varList_x++]= (p_tabElem);}
#endif /* CHECK_CNT */
15 // [re]initialize the varList
#define INIT_varList {varList_x= 0;}
#define var_CNT (varList_x)

// initialize all the variable-related space
#define INIT_vars {memset(varTable, 0,
20 VAR_TABLE_SIZE*sizeof(TabElem *)); INIT_varList; INIT_tabSpace;}
// to do something for each var in current constraint; sets p_tab_elem & its index
in varList
#define FOR_EACH_VAR(p_tab_elem, x) {int _i_var; for(_i_var= 0; ((x)= _i_var)<
var_CNT && ((p_tab_elem)= varList[_i_var]); _i_var++) {
25 #define END_FOR_EACH_VAR(p_tab_elem, x) }}

// buffer for return values
#define MAX_VAL_BUF_LEN (1024)
static Value valBuf[MAX_VAL_BUF_LEN];
static int valBuf_x= 0;
30 #ifdef CHECK_CNT
#define NEW_VALUE ((valBuf_x< MAX_VAL_BUF_LEN)?
&valBuf[valBuf_x++]: (valBuf_x= 0, printf("\n***valBuf-buffer overflow***\n"),
fflush(stdout), &valBuf[valBuf_x++]))
#else
35 #define NEW_VALUE ((valBuf_x< MAX_VAL_BUF_LEN)?
&valBuf[valBuf_x++]: (valBuf_x= 0, &valBuf[valBuf_x++]))
#endif /* CHECK_CNT */
#define INIT_valBuf {valBuf_x= 0;}

// buffer for enumerated-range terms
40 #define MAX_ENUM_RANGE_TERMS (128)
static P4TERM enumRangeTermBuf[MAX_ENUM_RANGE_TERMS]= {NULL};

```

```

static int enumRangeTermBuf_x= 0;
#define enumRangeTermBuf_CNT          (enumRangeTermBuf_x)
#define INIT_enumRangeTermBuf  {enumRangeTermBuf_x= 0;}
#ifdef CHECK_CNT
5  #define ADD_ENUM_RANGE_TERM(term)      {if (enumRangeTermBuf_x<
MAX_ENUM_RANGE_TERMS) enumRangeTermBuf[enumRangeTermBuf_x++]= (term);
else {printf("\n***enumRangeTermBuf overflow***\n");fflush(stdout);}}
#else
#define ADD_ENUM_RANGE_TERM(term)
10 {enumRangeTermBuf[enumRangeTermBuf_x++]= (term);}
#endif /* CHECK_CNT */
#define ENUM_RANGE_TERM(x)              (((x)< enumRangeTermBuf_CNT) &&
((x)>= 0)? enumRangeTermBuf[x]: NULL)
#define FOR_EACH_ENUM_RANGE_TERM(term, x)  {int _i_vrange; for(_i_vrange= 0;
15 (((x)= _i_vrange)< enumRangeTermBuf_x) && ((term)= enumRangeTermBuf[_i_vrange]);
_i_vrange++) {
#define END_FOR_EACH_ENUM_RANGE_TERM(term, x)  }}
// swap the positions of the terms (given by their indices in the buffer) in
the variable-range buffer
20 #define SWAP_ENUM_RANGE_TERMS(term_x, term_y)      {P4TERM _t_vrange;\
if (((term_x)< enumRangeTermBuf_CNT) && ((term_y)< enumRangeTermBuf_CNT)) {\
_t_vrange= enumRangeTermBuf[term_x]; enumRangeTermBuf[term_x]=
enumRangeTermBuf[term_y];\
enumRangeTermBuf[term_y]= _t_vrange;\
25 }}

// buffer for var-type (e.g. int(X); eq_vars(X, Y); neq_vars(X, Y,Z) ) terms
#define MAX_VAR_TYPES_TERMS          (128)
static P4TERM varTypesTermBuf[MAX_VAR_TYPES_TERMS]= {NULL};
static int varTypesTermBuf_x= 0;
30 #define varTypesTermBuf_CNT          (varTypesTermBuf_x)
#define INIT_varTypesTermBuf  {varTypesTermBuf_x= 0;}
#ifdef CHECK_CNT
#define ADD_VAR_TYPES_TERM(term) {if (varTypesTermBuf_x<
MAX_VAR_TYPES_TERMS) varTypesTermBuf[varTypesTermBuf_x++]= (term); else
35 {printf("\n***varTypesTermBuf overflow***\n");fflush(stdout);}}
#else
#define ADD_VAR_TYPES_TERM(term) {varTypesTermBuf[varTypesTermBuf_x++]=
(term);}
#endif /* CHECK_CNT */

40 // buffer for storing solutions so we can return solutions in some (e.g.
breadth-first) order
#define MAX_SOLN_BUF_LEN          (1024)
// the soln-buffer really is an array of vectors of solutions (e.g. [[x1,y1],[x2,y2],...]
```

```

//      (we take the vector-width to be the no. of variables in the constraint==
varList_x)
static Value solnBuf[MAX_SOLN_BUF_LEN];
static Value tmpsolnBuf[MAX_SOLN_BUF_LEN/2]; // tmp solution buffer - useful for
5  shuffling solutions
static int solnVec_x= 0, solnCnt= 0, curSoln_x= 0, doBufferSoln= FALSE;
#define INIT_solnBuf      {solnVec_x= 0; solnCnt= 0; curSoln_x= 0; doBufferSoln=
FALSE;}
#define NEW_SOLN_VECTOR      ((var_CNT<= 0)? NULL:\
10      ((solnVec_x<
(int)(MAX_SOLN_BUF_LEN/var_CNT))? &solnBuf[var_CNT*solnVec_x++]: NULL))
#define SOLN_VECTOR_SIZE      (var_CNT*sizeof(Value))
#define SET_BUF_SOLN_CNT(soln_cnt)      {solnCnt= (soln_cnt);}
// ptr to the value (by its index from the varList) of a variable in specified solution
15 (by its index) in the solnBuf
#define p_VAR_VALUE_in_solnBuf(soln_x, var_x)      (((soln_x)>= solnCnt)? NULL:
(((var_x)< var_CNT)? &solnBuf[var_CNT*(soln_x)+ (var_x)]: NULL))
// ptr to the value (by its index from the varList) of a variable in current-solution
#define CUR_BUF_SOLUTION(var_x)      p_VAR_VALUE_in_solnBuf(curSoln_x,
20 var_x)
#define NEXT_BUF_SOLUTION      ((curSoln_x>= (solnCnt-1))? NULL:
&solnBuf[varList_x*++curSoln_x])
#define FOR_EACH_SOLN_VECTOR(p_soln_vec, x)      {int _i_solnx; for(_i_solnx= 0;
(((x)= _i_solnx)< solnCnt) && ((p_soln_vec)= &solnBuf[var_CNT*_i_solnx]); _i_solnx++) {
25 #define END_FOR_EACH_SOLN_VECTOR(p_soln_vec, x)      }}
// swap the specified (by their indices) solutions (value-vectors) in the solnBuf
#define SWAP_SOLUTIONS(soln_i, soln_j)\
if ((soln_i)< solnCnt && (soln_j)< solnCnt)\
{\
30 Value __tmp; int __x;\
for(__x= 0; __x< var_CNT; __x++)\
{\
__tmp= *p_VAR_VALUE_in_solnBuf(soln_i, __x);\
*p_VAR_VALUE_in_solnBuf(soln_i, __x)=
35 *p_VAR_VALUE_in_solnBuf(soln_j, __x);\
*p_VAR_VALUE_in_solnBuf(soln_j, __x)= __tmp;\
}\
}
// move the specified (by its from & to-index) solution in the solnBuf
40 #define MOVE_SOLUTION(to_soln_x, from_soln_x)\
if ((to_soln_x)< solnCnt && (from_soln_x)< solnCnt)\
{\
int __x;\
for(__x= 0; __x< var_CNT; __x++)\
45 {\

```

```

        *p_VAR_VALUE_in_solnBuf(to_soln_x, __x)=
*p_VAR_VALUE_in_solnBuf(from_soln_x, __x);\
    }\
}
5      // copy the specified (by its index) solution from solnBuf into the tmpsolnBuf
#define COPY_SOLN_to_TMPBUF(soln_i)\
if ((soln_i)< solnCnt)\
    {\
    int __x;\
10    for(__x= 0; __x< var_CNT; __x++)\
        {\
            tmpsolnBuf[varList_x*(soln_i)+ __x]= *p_VAR_VALUE_in_solnBuf(soln_i,
__x);\
        }\
15    }
    // copy the specified (by its from & to-index) solution from tmpsolnBuf into the
solnBuf
#define COPY_SOLN_from_TMPBUF(to_soln_x, from_tmp_soln_x)\
if ((to_soln_x)< solnCnt && (from_tmp_soln_x)< solnCnt)\
20    {\
        int __x;\
        for(__x= 0; __x< var_CNT; __x++)\
            {\
                *p_VAR_VALUE_in_solnBuf(to_soln_x, __x)=
25    tmpsolnBuf[varList_x*(from_tmp_soln_x)+ __x];\
            }\
    }

    // misc. static var's
static int semError = 0; // set to non-zero in case of semantic error.
30 static BOOLEAN calledStartProlog4 = FALSE;
static BOOLEAN startedNewProlog4Session= FALSE;
static int resultFromProlog4= 0;
static BOOLEAN fractionalizeRational= FALSE; // if TRUE, fractionalize all rationals
static BOOLEAN roundoffFractionOnlyRational= TRUE; // if TRUE round-off fraction-only
35 rationals (e.g. 2/3); otherwise, return fraction-only rationals as fractions
static char *p4Argv[3]= {"P4LIB", "-banner=off", NULL}; // init-args to Prolog
IV
static BOOLEAN useIntervalSolver= FALSE;
static char curConstraint[MAX_EXPR_SIZE];
40 static double Precision= DEF_PRECISION;
static BOOLEAN RandomizeConstraints= FALSE;
static int SolnDiffWt = DEF_SOLN_DIFF_WT; // weight to indicate how
"different" the solns must be from each other

```

```

static long BSeed = 1;                                // seed for the random-bit-generator: brandom()
static BOOLEAN enumerateVarsRandomlyNoHistory = FALSE; // True if independent vars
are to be enumerated randomly (without keeping track of their past values)
static int TimedThreadCount= 0;    // keeps a running count of active threads
5 static HANDLE TimedSolnMutex; // a semaphore to access TimedThreadCount between
threads
static BOOLEAN AbortConstraintTimer= FALSE; // a flag to abort constraint-timer
static BOOLEAN AbortWatchPrologThread = FALSE; // a flag to abort WatchAbortPrologSoln
thread
10 static char *PrologSolnInterruptFile= NULL;    // Presence of this file indicates interruption
of Prolog-solution

        // misc. static declarations
static BOOLEAN init_solve_constraint(int keep_solns);
static TabElem *get_var(char *var);
15 static P4TERM get_var_term(char *var);
static Value *get_term_value(P4TERM term);
static List *cons(void *elem, List *lst);
static List *ncons(List *lst, void *elem);
static P4SYMBOL p4str2symbol(char *str); // pseudo p4-routine
20 static P4TERM p4make_atom_from_cstring(char *str); // pseudo p4-routine
static P4TERM P4Make_Rational(double val); // pseudo p4-routine
static int p4is_constant(P4TERM term);
static Value * get_var_value(char *var);
static P4TERM get_value_term(Value *val);
25 static int auxSolveConstraint(char *constraint, int keep_prev_soln, long msec);
static int h_auxSolveConstraint(char *constraint, int keep_prev_soln, int enum_vars_randomly,
long msec);
static P4TERM make_termlist2term(List *lst);
static P4TERM make_valslst2term(List *lst);
30 static P4TERM combine_terms_array(P4TERM terms[], long size_terms, int do_conjunct, int
randomize, P4TERM var, int var_eq);
static P4TERM combine_terms_list(List *terms_lst, int do_conjunct, int randomize, P4TERM
var, int var_eq);
static int auxSolveConstraintOrdered(char *constraint, int order_type, int max_soln);
35 static double align_val_with_precision(double val, double precision);
static TabElem *get_term_tabelem(P4TERM term);
static int getTimedThreadCount();

        // misc. extern declarations
extern int yyparse(void);
40 extern int yyerror(char *s);
extern int yylex(void);

```

```

#ifndef PRLGHLAPI
#include "cmn_drvr.c"           // driver to test all the stuff out
#endif /* PRLGHLAPI */

```

```

// public functions

```

```

5 char * CCONV GetHLAPIVersion()
  { // return the current version of the Prolog HL API
    return(VERSION);
  }

10 BSTR CCONV VBGetHLAPIVersion()    // wrapper to GetHLAPIVersion() for VB
  {
    char *c_ver;
    BSTR vb_ver;

    c_ver= GetHLAPIVersion();
    vb_ver= SysAllocStringByteLen(NULL, strlen(c_ver)+1); // alloc a new BSTR
15 strcpy((char *)vb_ver, c_ver);

    return(vb_ver);
  }

20 DWORD WINAPI watchAbortPrologSoln(void *null)
  { // watch AbortPrologSoln-variable or the presence-of-Interrupt-file; if either condition
    becomes true, then abort Prolog constraint-solving-process
    #define SLEEP_INTERVAL      (2000) // in msec
    extern int access();

    for(AbortWatchPrologThread= FALSE; !AbortWatchPrologThread;
    Sleep(SLEEP_INTERVAL)) // run until someone turns AbortWatchPrologThread to TRUE
25     {
        if (AbortPrologSoln || (PrologSolnInterruptFile && (access(PrologSolnInterruptFile,
        0)==0)))
            {
30                 prolog_events |= (1L<< 16);
            }
    }

    AbortWatchPrologThread= FALSE; // set if FALSE here as an indicator that this thread is
    finished

    return(0);
35 }

```

```

int CCONV SetPrologInterruptFile(char *interrupt_filename)
{
    // set the filename whose presence interrupts Prolog-solution
    PrologSolnInterruptFile= interrupt_filename? strdup(interrupt_filename): NULL;

    return(TRUE);
}

static int StartProlog4Session_aux(char *p4hlapilib_file, long heapsize, long choicesize)
{
    // starts Prolog IV, return true (1) if ok, false (0) otherwise.
    // p4hlapilib_file is the pathname to the high-level Prolog IV API library file
    // heapsize is the heap-stack size; choicesize is the choice-stack size.
    // ((argc, argv[]) are arguments to Prolog IV.)

    int status, argc;
    char *p4argv[32]= {"P4LIB", NULL};          // init-args to Prolog IV
    char *banner_arg= "-banner=off";
    char heapsize_str[32], choicesize_str[32];
    P4TERM term;
    long dummy, id;

    if (!(TimedSolnMutex= CreateMutex(NULL, FALSE, NULL))) // create a semaphore to
        access TimedThreadCount
        return(FALSE);

    AbortPrologSoln= FALSE;
    if (!CreateThread(NULL, 0, watchAbortPrologSoln, &dummy, 0, &id))
    {
        printf("Unable to create watch-thread\n");
        return(FALSE);
    }

    if (!calledStartProlog4)
    {
        calledStartProlog4 = TRUE;
        p4errno = 0;          // reset PrologIV error-no
        // format arguments for the P4 commandline

        argc= 1;
        p4argv[argc++]= banner_arg;
        p4argv[argc++]= "-heap";
        sprintf(heapsize_str, "%d", heapsize);
        p4argv[argc++]= heapsize_str;
        p4argv[argc++]= "-choice";
        sprintf(choicesize_str, "%d", choicesize);
        p4argv[argc++]= choicesize_str;

        p4argv[argc]= NULL;
    }
}

```

```

    status= (p4init(argc, p4argv)==0);
    if (!status)
        return(FALSE);

    srandom(time(NULL));
5    if (p4hlapilib_file && !Compile(p4hlapilib_file))           // load High-level
    PrologIV library
        return(FALSE);

        // set the randomizer-seed in Prolog - from time(NULL)
    p4new_session();
10    term= P4MAKE_FUNC_1("randomize", p4make_lint(time(NULL)));
    p4make_call(term);
    status = p4next_solution();
    if (status==P4SESSION_ERROR)
        return (FALSE);
15    p4finish_session();

#ifdef DO_SET_OF_INIT
    // we don't use setof/bagof for now - so, no need to do this initialization
    // initialization to get around a bug in P4 when calling setof/3 or bagof/3
    p4new_session();
20    term= P4AND(P4MAKE_FUNC_2("def_array",
    p4make_atom_from_cstring("tab_bagof"), p4make_lint(100)),
        P4AND(P4MAKE_FUNC_2("record",
    p4make_atom_from_cstring("block_limit"), p4make_lint(0)),
        P4MAKE_FUNC_2("record",
25    p4make_atom_from_cstring("last_bag_bound"), p4make_lint(0))));
    p4make_call(term);
    status = p4next_solution();
    p4finish_session();
    // end-of-initialization to get around a bug in P4 when calling setof/3 or bagof/3
30 #endif /* DO_SET_OF_INIT */
    }

    return(TRUE);
}

int CCONV StartProlog4Session(char *p4hlapilib_file)
35 {
    // starts Prolog IV, return true (1) if ok, false (0) otherwise.
    // p4hlapilib_file is the pathname to the high-level Prolog IV API library file

    return(StartProlog4Session_aux(p4hlapilib_file, P4HEAP_SIZE, P4CHOICE_SIZE));
}

```



```

int CCONV StartProlog4SessionSetStacks(char *p4hlapilib_file, long heapsize, long choicesize)
{
    // starts Prolog IV, return true (1) if ok, false (0) otherwise.
    // p4hlapilib_file is the pathname to the high-level Prolog IV API library file
    // heapsize is the heap-stack size; choicesize is the choice-stack size.

5   return(StartProlog4Session_aux(p4hlapilib_file, MAX(heapsize,P4HEAP_SIZE),
    MAX(choicesize, P4CHOICE_SIZE)));
}

int CCONV StartProlog4SessionDefault()
{
    // starts Prolog IV with default parameters, return true (1) if ok, false (0) otherwise.

10  return(StartProlog4Session_aux("c:\\MyDLL\\HLP4lib.p4", P4HEAP_SIZE,
    P4CHOICE_SIZE));
}

int CCONV EndProlog4Session()
{
    // end Prolog IV session; return true (1) if ok, false (0) otherwise.
15    // Abort all the threads: the WatchAbortPrologSoln, and constraint-timer
    AbortWatchPrologThread= TRUE;           // abort WatchAbortPrologSoln thread
    AbortConstraintTimer= TRUE;             // Abort all previous constraint-timers
    while(getTimedThreadCount()> 0) // Sleep until all constraint-timer threads are
    finished/aborted
20    Sleep(5);

    while(AbortWatchPrologThread) // Sleep until WatchAbortPrologSoln thread is
    finished/aborted
        Sleep(5);

    return(TRUE);
25 }

static char random()
// return a random-bit (0 or 1) (uses BSeed);
// uses primitive polynomial modulo 2 (PPM2) of degree 18
{
30 unsigned char newbit;

    newbit = (BSeed>> 17) & 1 // bit 18
            ^ (BSeed>> 4) & 1
            ^ (BSeed>> 1) & 1
            ^ (BSeed & 1);
35 BSeed = (BSeed<< 1) | newbit; // shift bits; put newbit at end

    return(newbit);

```

```

    }

    static double align_val_with_precision(double val, double precision)
    // align the given val with the given precision; return the aligned value.
    // (That is, returned-val= N* precision, (where N is integer) such that |returned-val - val| is
5    minimum
    {
        int N;

        if (precision== 0)
            return(val);

10    N= (val>= 0)? (int)((val/precision)+0.5): (int)((val/precision)- 0.5);
        val= N* precision;

        return(val);
    }

    // start-of Thread-related functions
15    static int incTimedThreadCount()
    {
        // increment the TimedThreadCount; (inside a semaphore because of multithreaded
        access.)
        WaitForSingleObject(TimedSolnMutex, INFINITE);
        TimedThreadCount++;
20    ReleaseMutex(TimedSolnMutex);

        return(0);
    }

    static int decTimedThreadCount()
    {
        // decrement the TimedThreadCount; (inside a semaphore because of multithreaded
25    access.)
        WaitForSingleObject(TimedSolnMutex, INFINITE);
        TimedThreadCount--;
        ReleaseMutex(TimedSolnMutex);

        return(0);
30    }

    static int getTimedThreadCount()
    {
        // return the current TimedThreadCount; (inside a semaphore because of multithreaded
        access.)
        int cnt;

35    WaitForSingleObject(TimedSolnMutex, INFINITE);
        cnt= TimedThreadCount;

```

```
ReleaseMutex(TimedSolnMutex);
```

```
return(cnt);  
}
```

```
DWORD WINAPI theConstraintTimer(void *p_msec)
```

```
{  
    long msec= *((long *)p_msec);
```

```
// printf("Going to sleep in the timer\n"); // -- debug
```

```
if (msec> 0)
```

```
{  
    incTimedThreadCount();    // increment the active-thread count  
    if (msec<= 1000)
```

```
        Sleep(msec);
```

```
    else
```

```
        {    // check AbortConstraintTimer frequently between sleeps  
            long t;  
            for(t=0; (t< msec) && !AbortConstraintTimer; t+= 1000)  
                Sleep(1000);  
        }
```

```
}
```

```
else
```

```
    return(0);
```

```
// printf("Out of sleep in the timer ... setting prolog_events\n"); // -- debug
```

```
prolog_events |= (1L<< 16);
```

```
decTimedThreadCount();    // decrement the active-thread count
```

```
return(0);  
}
```

```
    // end-of Thread-related functions
```

```
static int auxSolveConstraint(char *constraint, int keep_prev_solns, long msec)
```

```
{    // solve the given constraint (e.g. "X= Y+ 4, Y=2.");
```

```
    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
```

```
    // if keep_prev_solns is true, do not discard the previous solutions
```

```
    // if msec> 0, the solver exits in the given max. time (ms) - whether the constraint is  
    // solved or not
```

```
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
```

```
    // returns negative integer in error (e.g. if the constraint could not be parsed, or  
    // solver aborted).
```

```
int stat;
```

```

long id;

stat    = 0;
        // reset all Prolog flags/events
resultFromProlog4= 0;
5  AbortPrologSoln= FALSE;
prolog_events= 0;

if (msec> 0)
    {      // the solution-process must be time-limited
        AbortConstraintTimer= TRUE;      // Abort all previous constraint-timers
10     while(getTimedThreadCount()> 0)  // Sleep until all previous active threads are
finished/aborted
        Sleep(5);
        AbortConstraintTimer= FALSE;

        prolog_events= 0;      // reset all previous Prolog events
15     if (!CreateThread(NULL, 0, theConstraintTimer, &msec, 0, &id))
        {
            printf("Unable to create thread\n");
            return(ERR_UNABLE_TO_CREATE_THREAD);
        }
20     }

if (constraint)
    {      // a new constraint given
        semError= 0;
        if (startedNewProlog4Session) // a goal called previously - end that.
25         p4finish_session();
        // initialize all the tables, spaces, counts;
        if (!init_solve_constraint(keep_prev_solns))
            return(ERR_INITIALIZATION);
        // start a new Prolog session

30     p4new_session();
        startedNewProlog4Session= TRUE;

        if ((inExprBuf_cnt= strlen(constraint))> MAX_EXPR_SIZE)
            return(ERR_CONSTRAINT_TOO_LONG);

        strcpy(inExprBuf, constraint);
35     inExprBuf_x= 0;

        stat= yyparse();
    }

```

```
else if (calledStartProlog4) // backtrack over the previous result
```

```
{
    if (semError==0)
        resultFromProlog4= p4next_solution();
5      else
        return (FALSE);
}
```

```
if (p4errno!= 0)
```

```
10  {
    printf("\n***Unknown error from PrologIV in auxSolveConstraint()***\n");
    return (ERR_PARSE);
}
```

```
if (semError== 0 && stat== 0)
```

```
15  return((resultFromProlog4== P4SESSION_SOLUTION)? TRUE:
        (resultFromProlog4== P4SESSION_END)? FALSE:
        (prolog_events!= 0)? ERR_SOLN_INTERRUPTED:
ERR_PROLOG_SOLVER);
else if (semError!= 0)
    return(semError);
20  else
    return(ERR_PARSE);
}
```

```
static int h_auxSolveConstraint(char *constraint, int keep_prev_solns, int enum_vars_randomly,
long msec)
```

```
25  { // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
    needed
        // backtracks over the previous solution if the constraint is empty (i.e. NULL)
        // if keep_prev_solns is true, do not discard the previous solutions
        // if enum_vars_randomly is true, enumerate independent vars randomly, without
30  keepingtrack of their previous values
        // if msec> 0, the solver exits in the given max. time (ms) - whether the constraint is
        solved or not
        // >>(we LOSE track of unique solutions (-cnt) when
enum_vars_randomly is true.)<<
35  // return true (=1) [false (=0)] if the constraint is [un]solvable;
        // returns negative integer in error (e.g. if the constraint could not be parsed).
```

```
int stat;
```

```
if (constraint)
```

```
40  {
    if (strlen(constraint)> MAX_EXPR_SIZE)
        return(ERR_CONSTRAINT_TOO_LONG);
}
```

```

strcpy(curConstraint, constraint);
useIntervalSolver= FALSE;
enumerateVarsRandomlyNoHistory= enum_vars_randomly;

```

```

if ((stat=auxSolveConstraint(curConstraint, keep_prev_solns, msec))==0 || (stat> 0 &&
!IsFullyConstrained(NULL)))

```

```

{
    useIntervalSolver= TRUE;
    return(auxSolveConstraint(curConstraint, keep_prev_solns, msec));
}

```

```

return(stat);
}
else
return(auxSolveConstraint(NULL, keep_prev_solns, msec));
}

```

```

int CCONV SolveConstraint(char *constraint)
{
    // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
    needed

```

```

    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
    // returns negative integer in error (e.g. if the constraint could not be parsed).
return(h_auxSolveConstraint(constraint, FALSE, FALSE, -1));
}

```

```

int CCONV SolveConstraintRandomly(char *constraint)
{
    // random-solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver
    as needed

```

```

    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
    // >> NO track of unique solutions (-cnt) is kept : You may not get unique
    solutions <<
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
    // returns negative integer in error (e.g. if the constraint could not be parsed).

```

```

return(h_auxSolveConstraint(constraint, FALSE, TRUE, -1));
}

```

```

int CCONV TimedSolveConstraint(char *constraint, long msec)
{
    // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
    needed

```

```

    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
    // ensures that the call finishes in the given time (ms) - whether the constraint is solved or
    not
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
    // returns negative integer in error (e.g. if the constraint could not be parsed, or

```

could not be solved in given time).

```
return(h_auxSolveConstraint(constraint, FALSE, FALSE, msec));
}
```

```
int CCONV TimedSolveConstraintRandomly(char *constraint, long msec)
```

```
{ // random-solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver
as needed
```

```
    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
```

```
    // >> NO track of unique solutions (-cnt) is kept : You may not get unique
solutions <<
```

```
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
```

```
    // returns negative integer in error (e.g. if the constraint could not be parsed).
```

```
return(h_auxSolveConstraint(constraint, FALSE, TRUE, msec));
}
```

```
static int solution_cmp(const Value *vec1, const Value *vec2)
```

```
{ // compare the two solution-vectors; return >,< 0 as vec1 >,< vec2.
```

```
int i, wt, dist;
```

```
double val1, val2;
```

```
for(i= dist= 0; i< varList_x; i++)
```

```
{
```

```
    wt= varList_x- i;
```

```
    val1= (vec1[i].type== VAL_INTEGER)? vec1[i].value.integer: (vec1[i].type==
VAL_IRRATIONAL)? (vec1[i].value.real.lower.val+vec1[i].value.real.upper.val)/2:
```

```
        (vec1[i].type== VAL_REAL || vec1[i].type== VAL_RATIONAL_FLOAT ||
```

```
vec1[i].type== VAL_RATIONAL_FRACTION)? vec1[i].value.rational.real: 0;
```

```
    val2= (vec2[i].type== VAL_INTEGER)? vec2[i].value.integer: (vec2[i].type==
VAL_IRRATIONAL)? (vec2[i].value.real.lower.val+vec2[i].value.real.upper.val)/2:
```

```
        (vec2[i].type== VAL_REAL || vec2[i].type== VAL_RATIONAL_FLOAT ||
```

```
vec2[i].type== VAL_RATIONAL_FRACTION)? vec2[i].value.rational.real: 0;
```

```
    dist+= (int)(wt*(val1- val2)); // weighted distance
```

```
}
```

```
return(dist);
```

```
}
```

```
static int auxSolveConstraintOrdered(char *constraint, int order_type, int max_soln)
```

```
{ // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
needed
```

```
    // solve to find the given no. (= max_soln) of solutions if max_soln is positive; find
(nearly) all solutions if it is negative
```

```
    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
```

```

// Store all the solutions in a buffer, order them (by the given order_type)so we can return
solutions in an ordered fashion
// present the solutions conforming to the given order (e.g. ORDER_DIFF_TOGETHER)
// returns, on first call (i.e. when constraint is non-NULL), (1+
5 the_total_count_of_solutions) (> 0) [false (=0)] if the constraint is [un]solvable;
// (note that in case of constraints without variables (e.g. "4= 4."),
total-no.-of-solutions is 0, though the constraint is provable.)
// returns, on subsequent calls (i.e. when constraint is NULL), true (= 1) [false (=0)] if a
solution exists [does not exist];
10 // returns negative integer in error (e.g. if the constraint could not be parsed).
// (The P4 setof/3 & bagof/3 are buggy - hence we simulate them here ourselves.)
int i, iy, soln_cnt, half_cnt, stat;
Value *p_vec, *pval;
TabElem *p_tab_elem;
15 int keep_prev_solns;
extern void qsort();

keep_prev_solns= FALSE;
RandomizeConstraints= FALSE;
if (constraint)
20 {
for(soln_cnt= 0; ((max_soln< 0) || (soln_cnt< max_soln)) && ((stat=
h_auxSolveConstraint(constraint, keep_prev_solns, FALSE,-1))> 0); )
{
RandomizeConstraints= (order_type==ORDER_UNIQ_SOLUTIONS); //
25 randomize (if necessary) var-range-sequences on subsequent calls to solve the constraint
if (p_vec= NEW_SOLN_VECTOR)
{ // enough room in the soln-buffer exists -- get the solution-vector
for the variables
30 FOR_EACH_VAR(p_tab_elem, i)
if (pval= get_var_value(p_tab_elem->name))
p_vec[i]= *pval;
END_FOR_EACH_VAR(p_tab_elem, i)

++soln_cnt;
SET_BUF_SOLN_CNT(soln_cnt);
35 }
else // too many solutions (or, zero variables in constraint) to store
break;
constraint= (order_type==ORDER_UNIQ_SOLUTIONS)? constraint:NULL;
keep_prev_solns= (order_type==ORDER_UNIQ_SOLUTIONS);
40 }

doBufferSoln= TRUE;

```



```

if(soln_cnt<= 2)
    return(soln_cnt> 0? soln_cnt+1: (stat> 0)? 1: 0);

switch(order_type)    // -- order the solutions if so desired ----
{
5   case ORDER_LIKE_TOGETHER:    // try to gather like solutions together
        qsort(solnBuf, soln_cnt, sizeof(Value)*varList_x, solution_cmp);
        break;

        case ORDER_RANDOM:      // gather solutions in a random sequence - random
shuffle
        case ORDER_UNIQ_SOLUTIONS:
10         for(i= 0, half_cnt= soln_cnt/2; i< half_cnt; i++)
            {
                iy= half_cnt+ (random()%half_cnt);
                SWAP_SOLUTIONS(i, iy);
15             }
            break;

            case ORDER_DIFF_TOGETHER:    // try to gather "different" solutions
together - deterministic shuffle
                if(soln_cnt== 3)
                    { // just swap the last two elements
                        SWAP_SOLUTIONS(1, 2);
                        return(soln_cnt> 0);
                    }
                qsort(solnBuf, soln_cnt, sizeof(Value)*varList_x, solution_cmp);
25         for(i= 0, half_cnt= soln_cnt/2; i< half_cnt; i++) // intersperse the ordered
solutions
                    { // in preparation for shuffle, store the first half of the soln-vectors
in tmp-buffer
                        COPY_SOLN_to_TMPBUF(i);
30                     }
                    for(i= 0, half_cnt= soln_cnt/2; i< half_cnt; i++) // intersperse the ordered
solutions
                        { // shuffle by mapping first half as: index-> 2*index, and the
second half as: index ->2*(index- half_cnt)+1.
35                         MOVE_SOLUTION(2*i+1, i+ half_cnt);
                        COPY_SOLN_from_TMPBUF(2*i, i);
                        }
                        // after interspersing solutions, add some randomness to it too
40         for(i= 0, half_cnt= soln_cnt/2; i< half_cnt; i++)
            {
                if(brandom())
                    {

```

```

        iy= half_cnt+ (random()%half_cnt);
        SWAP_SOLUTIONS(i, iy);
    }
}

```

```

5         break;

```

```

        default:
            break;
    }

```

```

10     return(soln_cnt> 0? soln_cnt+1: (stat> 0)? 1: 0);

```

```

    }
else
    {
        return(NEXT_BUF_SOLUTION != NULL);
    }
15 }

```

```

int CCONV SolveConstraintOrdered(char *constraint, int order_type)
{
    // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
    // needed
    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
20    // present the (nearly) ALL the solutions conforming to the given order (e.g.
    ORDER_DIFF_TOGETHER)
    // returns, on first call (i.e. when constraint is non-NULL), (1+
    the_total_count_of_solutions) (> 0) [false (=0)] if the constraint is [un]solvable;
    // (note that in case of constraints without variables (e.g. "4= 4."),
25    total-no.-of-solutions is 0, though the constraint is provable.)
    // returns, on subsequent calls (i.e. when constraint is NULL), true (= 1) [false (=0)] if a
    solution exists [does not exist];
    // returns negative integer in error (e.g. if the constraint could not be parsed).
    return(auxSolveConstraintOrdered(constraint, order_type, -1));
30 }

```

```

int CCONV SolveConstraintOrderedNSolns(char *constraint, int order_type, int max_soln)
{
    // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as
    // needed
    // solve to find the given maximum no. (= max_soln) of solutions
35    // backtracks over the previous solution if the constraint is empty (i.e. NULL)
    // present the solutions conforming to the given order (e.g. ORDER_DIFF_TOGETHER)
    // returns, on first call (i.e. when constraint is non-NULL), (1+
    the_total_count_of_solutions) (> 0) [false (=0)] if the constraint is [un]solvable;
    // (note that in case of constraints without variables (e.g. "4= 4."),
40    total-no.-of-solutions is 0, though the constraint is provable.)
    // returns, on subsequent calls (i.e. when constraint is NULL), true (= 1) [false (=0)] if a

```

```

solution exists [does not exist];
    //      returns negative integer in error (e.g. if the constraint could not be parsed).
return(auxSolveConstraintOrdered(constraint, order_type, max_soln));
}

```

```

5  int CCONV SolveConstraintLin(char *constraint)
    {      // solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear solver only
          // backtracks over the previous solution if the constraint is empty (i.e. NULL)
          // return true (=1) [false (=0)] if the constraint is [un]solvable;
          //      returns negative integer in error (e.g. if the constraint could not be parsed).

```

```

10  useIntervalSolver= FALSE;
    return(auxSolveConstraint(constraint, FALSE, -1));
}

```

```

    int CCONV SetPrecision(double precision)
    { // set the precision for solving the constraint & for the solutions in the real domain
15    // returns TRUE if ok
      if (precision<= 0)
        return(FALSE);
      Precision= precision;

      return(TRUE);
20  }

```

```

    int CCONV SetSolnDiffWt(int soln_diff_wt)
    { // set the weight to indicate how "different" the solutions must be from each other in
      Uniq_Soln_Order
        //      (the higher the weight, the more the solutions are "different".)
25    // returns TRUE if ok
      if (soln_diff_wt< 0)
        return(FALSE);
      SolnDiffWt= soln_diff_wt;

      return(TRUE);
30  }

```

```

    int CCONV FractionalizeRational(int do_fractionalize)
    { // fractionalize all the rationals if do_fractionalize is TRUE; not otherwise (fractionalization
      may slow things down a bit.)
      fractionalizeRational= do_fractionalize;
35  return(TRUE);
    }

```

```

    int CCONV RoundoffFractionOnlyRational(int do_roundoff)

```

```

{ // roundoff fraction-only rationals (e.g. 2/3) if do_roundoff is TRUE; otherwise, return
fraction-only rationals as fractions
roundoffFractionOnlyRational= do_roundoff;
return(TRUE);
}

```

```

int CCONV IsIndependentVar(char *var)
{ // return TRUE (1) if the given variable is independent (i.e. specified in an enumeration);
FALSE (0) otherwise
TabElem *p_tab_elem;

```

```

if (var && (p_tab_elem= get_var(var)))
    return(p_tab_elem->is_independent_var);
else
    return(FALSE);
}

```

```

Value * CCONV GetValue(char *var)
{ // return the ptr to the value (in Value structure) of the given variable (e.g. "Area") if
known;
// return NULL on error (e.g. unknown variable, or variabe has no value, ...)
int i;

```

```

Value *val;
TabElem *p_tab_elem;

val= NULL;

```

```

if (!doBufferSoln)
{
    val= get_var_value(var);
}
else
{ // retrieve appropriate value from the soln-buffer
FOR_EACH_VAR(p_tab_elem, i)
    if (!strcmp(var, p_tab_elem->name))
    { // found the var in varList
        val= CUR_BUF_SOLUTION(i);
        break;
    }
}
END_FOR_EACH_VAR(p_tab_elem, i)
}

```

```

return(val);
}

```

```

long CCONV GetValue_type(Value *val) // return type (e.g. VAL_INTEGER) of the given
Value;
{
// returns VAL_UNKNOWN in
5 error
return(val? val->type: VAL_UNKNOWN);
}

```

```

long CCONV GetVarValue_type(char *var) // return type (e.g. VAL_INTEGER) of the given
variable;
10 {
// returns VAL_UNKNOWN in
error
Value *val;

return ((val= GetValue(var))? val->type: VAL_UNKNOWN);
15 }

```

```

long CCONV GetValue_int(Value *val) // return integer value of the given Value structure;
{
// return
ERR_GETVALUE_INT in error (e.g. given structure is not integer)
return((val && val->type== VAL_INTEGER)? val->value.integer: ERR_GETVALUE_INT);
20 }

```

```

Rational CCONV GetValue_rational(Value *val) // return rational value of the given Value
structure;
{
// return <ERR_GETVALUE_RAT,
ERR_GETVALUE_INT, ERR_GETVALUE_INT> in error (e.g. given structure is not rational)
25 Rational rat;

```

```

if (val && val->type== VAL_RATIONAL_FLOAT || val && val->type==
VAL_RATIONAL_FRACTION)
return(val->value.rational);
else
30 {
rat.real= ERR_GETVALUE_RAT;
rat.num = rat.den = ERR_GETVALUE_INT;
return(rat);
}
35 }

```

```

double CCONV GetValue_rational_float(Value *val) // return float rep. of the given
rational Value
{
return((val && (val->type== VAL_RATIONAL_FLOAT || val->type==

```

```

VAL_RATIONAL_FRACTION))? val->value.rational.real: ERR_GETVALUE_RAT);
}

```

```

long CCONV GetValue_rational_numer(Value *val)
{
    // return numerator of the fractional rep. of the given rational Value
5 return((val && val->type== VAL_RATIONAL_FRACTION)? val->value.rational.num: 0);
}

```

```

long CCONV GetValue_rational_denom(Value *val)
{
    // return denominator of the fractional rep. of the given rational Value
10 return((val && val->type== VAL_RATIONAL_FRACTION)? val->value.rational.den: 0);
}

```

```

Real CCONV GetValue_real(Value *val) // return real value (i.e. lower & upper bound) from
the given non-rational Value structure;
{
    // return <1, 0> in error (e.g. given structure is not
15 real)
Real rl;

```

```

if (val && val->type== VAL_IRRATIONAL)
    return(val->value.real);
else
{
    // error
20 rl.lower.val= 1;
    rl.upper.val= 0;
    rl.lower.is_infinite= rl.upper.is_infinite= TRUE;
    return(rl);
}
25 }

```

```

double CCONV GetValue_real_lower(Value *val) // return lower bound for the given
non-rational real value
{
    // return ERR_GETVALUE_REAL in error
30 return(val && val->type== VAL_IRRATIONAL?
    (val->value.real.lower.is_infinite? ERR_GETVALUE_REAL:
    val->value.real.lower.val): ERR_GETVALUE_REAL);
}

```

```

double CCONV GetValue_real_upper(Value *val) // return upper bound for the given
non-rational real value
35 {
    // return ERR_GETVALUE_REAL in error
return(val && val->type== VAL_IRRATIONAL?
    (val->value.real.lower.is_infinite? ERR_GETVALUE_REAL:
    val->value.real.upper.val): ERR_GETVALUE_REAL);
}

```

```

BSTR CCONV VBGetValue_string(Value *val)    // VB wrapper for GetValue_string()
{
    char *c_val;
    BSTR vb_val;

5    if (!val)
        return (NULL);
    vb_val= NULL;
    if (c_val= GetValue_string(val))
    {
10        vb_val= SysAllocStringByteLen(NULL, strlen(c_val)+1);    // alloc a new BSTR
        strcpy((char *)vb_val, c_val);
    }

    return(vb_val);
}

15    BSTR CCONV VBGetVarValue(char *var) // VB wrapper for GetVarValue()
    {
        char *c_val;
        BSTR vb_val;

        if (!var)
20            return (NULL);
        vb_val= NULL;
        if (c_val= GetVarValue(var))
        {
25            vb_val= SysAllocStringByteLen(NULL, strlen(c_val)+1);    // alloc a new BSTR
            strcpy((char *)vb_val, c_val);
        }

        return(vb_val);
    }

char * CCONV GetVarValue(char *var)    // return (uniform) string representation of
30    the given variable    //      returns NULL in error
    {
        Value *val;

        if (!var)
            return (NULL);

35    return ((val= GetValue(var))? GetValue_string(val): NULL);
    }

```

```

int CCONV GetVarValueBuf(char *var, int valuebuf_len, char valuebuf[])           //
return (uniform) string representation of the given variable in the value-buffer
{
    // returns length (>0) of
    the value in the valuebuf; returns <= 0 in error

```

```

5 Value *val;
  char *val_str;
  int val_len;

```

```

  if (!var || !valuebuf || valuebuf_len <= 0)
      return (-1);

```

```

10  if (valuebuf_len > 0)
      valuebuf[0] = 0;
  val_str = (val = GetValue(var)) ? GetValue_string(val) : NULL;
  val_len = val_str ? strlen(val_str) : 0;
  if ((val_len > 0) && (valuebuf_len > val_len))

```

```

15  {
      strcpy(valuebuf, val_str);
      free(val_str);
  }

```

```

  else
      return (-1);

```

```

  return (val_len);
}

```

```

char * CCONV GetValue_string(Value *val)           // return (uniform) string representation of
the given Value structure

```

```

25 { // return NULL in error (e.g. given structure is not valid)
  char str[1024], *p_tmp;
  List *lst;

```

```

  if (!val)
      return(NULL);

```

```

30  switch(val->type)
  {
      case VAL_INTEGER:
          sprintf(str, "%ld", val->value.integer);
          break;

```

```

35  case VAL_RATIONAL_FLOAT:
          sprintf(str, "%f", val->value.rational.real);
          break;

```

```

  case VAL_RATIONAL_FRACTION:
      if (val->value.rational.den != 1)

```



```

    sprintf(str, "%d/%d", val->value.rational.num, val->value.rational.den);
else
    sprintf(str, "%d", val->value.rational.num);
break;

```

```

5  case VAL_IRRATIONAL:
    if (val->value.real.lower.is_infinite)
        sprintf(str, "<--, ");
    else
        sprintf(str, "(%f, ", val->value.real.lower.val);
10  if (val->value.real.upper.is_infinite)
        sprintf(str+strlen(str), "-->");
    else
        sprintf(str+strlen(str), "%f)", val->value.real.upper.val);
    break;

```

```

15  case VAL_VAR:
        strcpy(str, "_");
        break;

```

```

        case VAL_STRING:
            strcpy(str, val->value.string);
20  break;

```

```

        case VAL_FUNCTOR:
            sprintf(str, "%s/%d", val->value.functor.predicate, val->value.functor.arity);
            break;

```

```

        case VAL_LIST:
25  strcpy(str, "[");
        for(lst= val->value.list; lst; lst= lst->next)
        {
            if (p_tmp= GetValue_string((Value *)lst->elem))
                strcat(str, p_tmp);
30  if (lst->next)
                strcat(str, ", ");
        }
        strcat(str, "]");
        break;

```

```

35  default:
            return(NULL);
        }

```

```

return(strdup(str));

```

```

}

int CCONV IsFullyConstrained(char *constraint)
{ // returns TRUE if the given constraint is fully constrained (i.e. solvable & all variables are
  constant); FALSE otherwise (or in error)
5   // (Check the immediately previous constraint if the given constraint is NULL.)
      // Note: It works only for the linear constraints because we use the linear-solver only here.
      //           As such, it labels nonlinear constraints such as:  $Y^2 = 2$ . as
unconstrained.
      //           The trouble with using interval-solver is that it may take a long
10   time
      //           to solve unconstrained equations e.g.  $X = Y + 2$ . (because it finds
all the solutions at once.)
int i;
Value *pval;
15   TabElem *p_tab_elem;

if (!constraint || SolveConstraintLin(constraint) > 0)
{
  FOR_EACH_VAR(p_tab_elem, i)
    if (pval = doBufferSoln? CUR_BUF_SOLUTION(i): get_term_value(p_tab_elem->term))
20     { // ?? should we use get_var_value() here instead of get_term_value() ???
        if (pval->type == VAL_VAR ||
            (pval->type == VAL_IRRATIONAL &&
             (pval->value.real.lower.is_infinite || pval->value.real.upper.is_infinite &&
              ABS(pval->value.real.upper.val - pval->value.real.lower.val) >
25   p_tab_elem->precision)))
            break;
        }
    END_FOR_EACH_VAR(p_tab_elem, i)
    return(i >= var_CNT); // true if all var's are constant
30 }
else
  return(FALSE);
}

BSTR CCONV VBPrintAllVarVals() // VB wrapper (almost) for PrintAllVarVals()
35 {
  char *c_val, buf[1024];
  BSTR vb_val;

  vb_val = NULL;
  if (c_val = PrintAllVarVals(buf))
40   {
    vb_val = SysAllocStringByteLen(NULL, strlen(c_val)+1); // alloc a new BSTR

```

```

        strcpy((char *)vb_val, c_val);
    }

    return(vb_val);
}

5 char * CCONV PrintAllVarVals(char buf[])
{ // Print all the var's with their values in the given buffer; return ptr to the given buffer
  // (assumes buffer is large enough to store all the var's.)
  int i;
  TabElem *p_tab_elem;
10 char tmp[256];

  if (!buf)
    return(NULL);
  buf[0]= 0;
  FOR_EACH_VAR(p_tab_elem, i)
15   sprintf(tmp, "%s: %s, ", p_tab_elem->name, GetValue_string(GetValue(p_tab_elem->name)));
   strcat(buf, tmp);
  END_FOR_EACH_VAR(p_tab_elem, i)
  if (strlen(buf)>= 2)
    buf[strlen(buf)-2]= 0;      // remove the last comma

20 return(buf);
}

char * CCONV PrintAllVarValsAllocate()
{ // Print all the var's with their values in an allocated buffer; return ptr to the given buffer
  char buf[2048];

25 PrintAllVarVals(buf);

  return(strdup(buf));
}

int CCONV Compile(char *p4_filename)  // compile & load the given p4_filename
(containing Prolog IV program)
30 { // return true (1) if the file compiled ok, false (0) in error
  int stat;

  p4new_session();
  p4make_call(P4MAKE_FUNC_1("compile", p4make_atom_from_cstring(p4_filename)));
  stat= (p4next_solution()== P4SESSION_SOLUTION);
35 p4finish_session();

```

```

return(stat);
}

```

```

static int term_numden(P4TERM term, int *num, int *den)
{ // performs fractional representation for the given rational term i.e. rational= num/den (e.g. 4.5=
9/2)

```

```

    // uses numden/3 to achieve that. (note that you can call a Prolog predicate in the middle of
another - in a recursive fashion.)

```

```

    // puts the numerator in the given num, and denominator in den.

```

```

    // returns TRUE if succeeded in doing the transformation, FALSE otherwise.

```

```

int stat;

```

```

P4TERM tnum, tden;

```

```

if (!term)

```

```

    return(FALSE);

```

```

*num= *den= 0;

```

```

p4new_session(); // note that we start a session before making new terms - these new terms are
valid only for this session

```

```

tnum= check_term(p4make_var());

```

```

tden= check_term(p4make_var());

```

```

p4make_call(P4MAKE_FUNC_3("numden", term, tnum, tden));

```

```

if ((stat= (p4next_solution()== P4SESSION_SOLUTION)) && p4is_integer(tnum) &&
p4is_integer(tden))

```

```

{

```

```

    *num= p4val_lint(tnum);

```

```

    *den= p4val_lint(tden);

```

```

}

```

```

p4finish_session();

```

```

return(stat);

```

```

}

```

```

static int numden(double rational, int *num, int *den)

```

```

{ // performs fractional representation for the given rational number i.e. rational= num/den (e.g.
4.5= 9/2)

```

```

    // uses numden/3 to achieve that. (note that you can call a Prolog predicate in the middle of
another - in a recursive fashion.)

```

```

    // puts the numerator in the given num, and denominator in den.

```

```

    // returns TRUE if succeeded in doing the transformation, FALSE otherwise.

```

```

return(term_numden(check_term(P4Make_Rational(rational)), num, den));
}

```

```

static int gcd(int x, int y)
{ // return the GCD of the two given integers
int num, den, tmp;

if (x<y)
5   { // swap x & y
    tmp= x;
    x= y;
    y= x;
  }
10  if (x== 0 || y== 0 || x==1 || y==1)
    return(1);
    if (x==y || x==(-y))
      return(x);

if (numden((((double)x)/y, &num, &den))
15  return((int)(x/num));
    else
      return(1);
  }

// private functions
20  int GetInString(char buf[], int max_size) // used in lexer - make it static later
    { // put input string (upto max_size) in the given buf[]; (usually called by YY_INPUT from
      lexical analyzer)
      // return the no. of char's actually put in buf[]. (returns 0 if no char's put.)
      int out_cnt;

25  out_cnt= MIN(max_size, (inExprBuf_cnt- inExprBuf_x)); // max. no. of char's to put in buf[]
      memcpy(buf, inExprBuf+inExprBuf_x, out_cnt);
      inExprBuf_x += out_cnt;

      return(out_cnt);
    }

30  static int bounds_of_real_var(P4TERM var, Real *realp)
    { // find the lower & upper bounds of the given (numeric) (real) var;
      // return true (1) on success; false (0) on failure (e.g. given var is not numeric)
      // (much of this func suggested by Pascal Bouvier of Prologianet)
      // (we use glb/2 & lub/2 instead of bounds/3 so we can assign the
35  // infinite-bound to the appropriate end (i.e. lower/upper))
      P4TERM tA, tB;
      int done, status;

```

```

done= FALSE;
realp->lower.is_infinite= realp->upper.is_infinite= FALSE;
realp->lower.val= realp->upper.val= 0;

```

```

// well, for efficiency, let's use bounds/3 to check if the var is unbounded on either side

```

```

5  p4new_session();
   tA = p4make_var();
   tB = p4make_var();
   p4make_call(P4MAKE_FUNC_3("bounds", var, tA, tB));
   if ((status= p4next_solution())!= P4SESSION_SOLUTION)
10      {
         if (status != P4SESSION_ERROR)
            realp->lower.is_infinite= realp->upper.is_infinite= TRUE; // <<-- KNJ: This is
not strictly correct, REVISIT it later -->>
         else
15            {printf("\n***Encountered error 1 in
bounds_of_real_var()***\n");fflush(stdout);}
            done= TRUE;
        }
   p4finish_session();
20  if (done)
       return(TRUE);

```

```

// first, check if the given var. is really unbounded on the lower side

```

```

p4new_session();
p4make_call(P4MAKE_FUNC_2("lt", var, P4Make_Rational(DEF_LOWER_BOUND)));
25  if ((status= p4next_solution())== P4SESSION_SOLUTION)
        {
            realp->lower.is_infinite= TRUE;
        }
   else
30      {
         // not unbounded on the lower side - find the actual lower bound
         if (status== P4SESSION_ERROR)
            {printf("\n***Encountered error 2 in
bounds_of_real_var()***\n");fflush(stdout);}

```

```

         p4finish_session();
35      p4new_session();
         tA = p4make_var();
         p4make_call(P4MAKE_FUNC_2("glb", var, tA)); // get the lower bound
         switch(p4next_solution())
            {
40              case P4SESSION_END:          realp->lower.is_infinite= TRUE; break; // no
solution - the bound is infinite
              case P4SESSION_SOLUTION:      realp->lower.val= p4val_rational(tA);

```

```

SAME(realp->lower.val, DEF_LOWER_BOUND) ||
realp->lower.is_infinite=
5  DEF_LOWER_BOUND);
                                   (realp->lower.val<
                                   break;
                                   // error - given var may be
                                   case P4SESSION_ERROR:
non-numeric
                                   default:      return(FALSE);
10      }
      }
      p4finish_session();

      // next, check if the given var. is really unbounded on the upper side
      p4new_session();
15  p4make_call(P4MAKE_FUNC_2("gt", var, P4Make_Rational(DEF_UPPER_BOUND)));
      if ((status= p4next_solution())== P4SESSION_SOLUTION)
      {
          realp->upper.is_infinite= TRUE;
      }
20  else
      {
          // not unbounded on the upper side - find the actual upper bound
          if (status== P4SESSION_ERROR)
              {printf("\n***Encountered error 3 in
bounds_of_real_var()***\n");fflush(stdout);}

          p4finish_session();
          p4new_session();
          tA = p4make_var();
          p4make_call(P4MAKE_FUNC_2("lub", var, tA)); // get the upper bound
          switch(p4next_solution())
30      {
          case P4SESSION_END:      realp->upper.is_infinite= TRUE; break; // no
solution - the bound is infinite
          case P4SESSION_SOLUTION: realp->upper.val= p4val_rational(tA);
                                   realp->upper.is_infinite=
35  SAME(realp->upper.val, DEF_UPPER_BOUND) ||
                                   (realp->lower.val>
                                   DEF_UPPER_BOUND);
                                   break;
40      case P4SESSION_ERROR:      // error - given var may be
non-numeric
                                   default:      return(FALSE);
                                   }

```

```

    }
    p4finish_session();

    return(TRUE);
}

```

```

5  static P4TERM get_value_term(Value *val)
    { // return the Prolog IV term for the given basic value (in Value struct)
      if (!val)
        return(NULL);
      switch(val->type)
10     {
        case VAL_INTEGER: return(check_term(p4make_lint(val->value.integer))); break;
        case VAL_RATIONAL_FLOAT:
          return(check_term(P4Make_Rational(val->value.rational.real))); break;
        case VAL_RATIONAL_FRACTION:
15         return(((val->value.rational.den != 1) && (val->value.rational.den != 0)) ?

          check_term(P4Make_Rational(((double)val->value.rational.num/((double)val->value.rational.den)
          )):
            check_term(P4Make_Rational((double)val->value.rational.num)));
20         break;
        case VAL_IRRATIONAL: // ?? what is the P4-call to make a P4term for an irrational
          ???
            return(NULL); break;
        case VAL_VAR: return(NULL); break;
25     case VAL_STRING:
          return(check_term(p4make_atom_from_cstring(val->value.string))); break;
        case VAL_FUNCTOR: // ?? what is the P4-call to make a P4term for a functor ???
          return(NULL); break;
        case VAL_LIST: return(check_term(make_valslst2term(val->value.list))); break;
30     default: return(NULL);
    }

    return(NULL);
}

```

```

static Value * get_var_value(char *var)
35 { // return the (ptr to) value (in Value struct) of the given PrologIV variable
    // (It finds the basic value through get_term_value(), and
    // then interprets that further for the given var & its type (e.g. precision/type considerations)
    )
    TabElem *p_tab_elem;
40 Value *val;

```



```

if (!var)
    return(NULL);

if (!(p_tab_elem= get_var(var)) || !(val= get_term_value((P4TERM)p_tab_elem->term)))
{
5     if (!val)
        {printf("\n***Recvd NULL value for the given variable %s;***\n",
var);fflush(stdout);}
        return(NULL);
    }
10    switch(val->type)
    {
        case VAL_VAR:
            /*      <-- KNJ: ignore all this for now - for efficiency's sake -- REVISIT it later
-->>
15        if (useIntervalSolver && bounds_of_real_var((P4TERM)p_tab_elem->term,
&val->value.real))
            {
                val->type= VAL_IRRATIONAL;
                if (!val->value.real.lower.is_infinite &&
20                !val->value.real.upper.is_infinite &&
                    ABS(val->value.real.upper.val- val->value.real.lower.val)<=
p_tab_elem->precision)
                        { // the range is smaller than the variable's precision - it can be represented
as a rational
25                val->type= VAL_RATIONAL_FLOAT;
                    val->value.rational.real= (val->value.real.lower.val+
val->value.real.upper.val)/2;
                    val->value.rational.num= val->value.rational.den= 0;
                    if (p_tab_elem->type== VAL_RATIONAL_FRACTION)
30                    {
                        numden(val->value.rational.real, &val->value.rational.num,
&val->value.rational.den);
                        val->type= VAL_RATIONAL_FRACTION;
                    }
35                }
            }
        /*
        break;

        case VAL_IRRATIONAL:
40    #ifdef OLD_IRRATIONAL
        if (!val->value.real.lower.is_infinite &&
            !val->value.real.upper.is_infinite &&
            ABS(val->value.real.upper.val- val->value.real.lower.val)<=

```

```

p_tab_elem->precision)
    { // the range is smaller than the variable's precision - it can be represented as a rational
        // this introduces errors (due to roundoff) which, however small, are
        unacceptable for ETS work

```

```

5      val->type= VAL_RATIONAL_FLOAT;
      val->value.rational.real= align_val_with_precision((val->value.real.lower.val+
val->value.real.upper.val)/2, p_tab_elem->precision);
      val->value.rational.num= val->value.rational.den= 0;
      if (p_tab_elem->type== VAL_RATIONAL_FRACTION)

```

```

10         {
            numden(val->value.rational.real, &val->value.rational.num,
&val->value.rational.den);
            val->type= VAL_RATIONAL_FRACTION;
        }

```

```

15     }
#endif /* OLD_IRRATIONAL */
    break;

```

```

    case VAL_RATIONAL_FLOAT:
        val->value.rational.real= align_val_with_precision(val->value.rational.real,
20    p_tab_elem->precision);
        if (p_tab_elem->type== VAL_RATIONAL_FRACTION)
            {
                numden(val->value.rational.real, &val->value.rational.num, &val->value.rational.den);
                val->type= VAL_RATIONAL_FRACTION;
25            }
        break;

```

```

    default:
        break;
    }

```

```

30    return(val);
}

```

```

static int are_equal_terms(P4TERM term1, P4TERM term2)
{ // returns true if the two given terms are equal
int stat;

```

```

35    if (!term1 || !term2)
        return(FALSE);

```

```

    p4new_session();
    p4make_call(P4EQ(term1, term2));
    stat= (p4next_solution()== P4SESSION_SOLUTION);

```

```
p4finish_session();
```

```
return(stat);  
}
```

```
static Value *get_term_value(P4TERM term)
```

```
{ // return the basic value (in Value struct) of the given Prolog IV term
```

```
Value *val, *p_tmp_val;
```

```
int term_type;
```

```
if (!term || p4is_nil(term))
```

```
return(NULL);
```

```
10 //val= (Value *)calloc(1, sizeof(Value));
```

```
val= NEW_VALUE;
```

```
switch(term_type= P4WHAT_IS(term))
```

```
{
```

```
case P4INTEGER:
```

```
15 val->type= VAL_INTEGER;
```

```
val->value.integer= p4val_int(term);
```

```
break;
```

```
case P4FLOAT:
```

```
val->type= VAL_RATIONAL_FLOAT;
```

```
20 val->value.rational.real= p4val_double(term);
```

```
val->value.rational.num= val->value.rational.den= 0;
```

```
break;
```

```
case P4RATIONAL:
```

```
val->type= VAL_RATIONAL_FLOAT;
```

```
25 val->value.rational.real= p4val_rational(term);
```

```
if (!roundoffFractionOnlyRational && !are_equal_terms(term,  
check_term(P4Make_Rational(val->value.rational.real))))
```

```
{ // if the rational cannot be represented in non-fractional terms
```

```
val->type= VAL_RATIONAL_FRACTION;
```

```
30 term_numden(term, &val->value.rational.num, &val->value.rational.den);
```

```
}
```

```
break;
```

```
case P4ATOM:
```

```
val->type= VAL_STRING;
```

```
35 val->value.string= p4val_cstring(term); // do we need to realloc the string ?
```

```
break;
```

```
case P4DOT:
```

```

        val->type= VAL_LIST;
        p_tmp_val= get_term_value(p4cdr(term));
        val->value.list= cons(get_term_value(p4car(term)), p_tmp_val? p_tmp_val->value.list:
5      NULL);
        break;

```

```

    case P4FUNCTOR:
        val->type= VAL_FUNCTOR;
        val->value.functor.predicate= p4val_cstring(term);
        val->value.functor.arity= p4get_arity(term);
10      break;

```

```

    case P4VAR:
        val->type= VAL_VAR;
        if (bounds_of_real_var(term, &val->value.real) && // <<-- added check: KNJ
            !val->value.real.lower.is_infinite && !val->value.real.upper.is_infinite)
15      val->type= VAL_IRRATIONAL;
        break;

```

```

    default:
        printf("\n***Recvd unknown value-type for the given variable;***\n");
        fflush(stdout);
20      //free(val);
        return(NULL);
    }

```

```

    if (p4errno!= 0)
        {printf("\n***Unknown error occurred when getting value of a
25      variable;***\n");fflush(stdout);}

    return(val);
}

```

```

        // aux functions
30      static int hash(char *str)
        {
            int i, h;

            for(i= h= 0; str[i]; i++) h += str[i]*i;

            return(h% VAR_TABLE_SIZE);
35      }

```

```

static TabElem *get_var(char *var)
{    // ptr to the appropriate variable-entry in the var-table

```

```
TabElem *p;
```

```
for(p= varTable[hash(var)]; p && p->name && strcmp(p->name, var,  
MAX_VAR_NAME_LEN); p= p->next);  
return(p);  
}
```

```
static P4TERM get_var_term(char *var)  
{ // return the term associated with the variable  
TabElem *p;
```

```
p= get_var(var);  
return(p? (P4TERM)p->term: NULL);  
}
```

```
static TabElem *get_term_tabelem(P4TERM term)  
{ // return the var-table-element associated with the given term  
int i;  
TabElem *p_tab_elem;
```

```
FOR_EACH_VAR(p_tab_elem, i)  
if (p_tab_elem->term== term)  
return(p_tab_elem);  
END_FOR_EACH_VAR(p_tab_elem, i)
```

```
return(NULL);  
}
```

```
static int set_term_on_grid(P4TERM term)  
{  
TabElem *p_tab_elem;  
if (p_tab_elem= get_term_tabelem(term))  
p_tab_elem->ongrid= TRUE;
```

```
return TRUE;  
}
```

```
static int reset_term_on_grid(P4TERM term)  
{  
TabElem *p_tab_elem;  
if (p_tab_elem= get_term_tabelem(term))  
p_tab_elem->ongrid= FALSE;
```

```
return TRUE;  
}
```

```

static void insert_const(P4TERM term, Value *const_value)
{      // insert the given {term, constant value} pair in constBuf[]
ADD_constBuf(term, *const_value);
return;
}

```

```

static Value *get_const_value(P4TERM term)
{      // return the Value associated with the given term if the term is numeric-constant; NULL
otherwise.
int i;

```

```

for(i= 0; (i< constBuf_x) && (constBuf[i].term != term); i++);
return((i< constBuf_x)? &constBuf[i].value: NULL);
}

```

```

static P4TERM insert_var(char *var)
{ // insert given var (with an associated Prolog term) to the var-table if not already present
//      return ptr to the term associated with the var.
TabElem *p;
int h;

```

```

if (!strcmp(var, "_"))
{      // anonymous variable - just return a Term for it
return(check_term(p4make_var()));
}
else if (!(p= get_var(var)))
{      // non-anonymous variable - store in variable table
p= NEW_TAB_ELEM;
// initialize variable to default type
p->type= DEF_VAR_TYPE; // <-- actual type (e.g. int, ...) may be put later
p->is_independent_var = FALSE;
p->precision= DEF_PRECISION;
p->ongrid= DEF_GRID;
strcpy(p->name, var, MAX_VAR_NAME_LEN);
p->term= check_term(p4make_var());
p->next= varTable[h= hash(var)];
varTable[h]= p;
ADD_varList(p);
}
return((P4TERM)p->term);
}

```

```

static P4TERM insert_var_with_precision(char *var, double precision)
{ // insert (or modify its attribs if preexistent) given var (with an associated Prolog term) to the
var-table;

```

```

        //      (attach the given precision to the variable)
        //      return ptr to the term associated with the var.
TabElem *p;
int h;

5  if (!(p= get_var(var)))
    {
        p= NEW_TAB_ELEM;
        // initialize variable to default type
        p->type= DEF_VAR_TYPE; // <-- actual type (e.g. int, ...) may be put later
10     p->is_independent_var = FALSE;
        p->ongrid= DEF_GRID;
        strncpy(p->name, var, MAX_VAR_NAME_LEN);
        p->term= check_term(p4make_var());
        p->next= varTable[h= hash(var)];
15     varTable[h]= p;
        ADD_varList(p);
    }

    p->precision= precision;

    return((P4TERM)p->term);
20 }

static int mark_var_type(char *var, int type)
{
    // mark the type of the given variable; return TRUE if ok, FALSE in error
    TabElem *p;

    if (!(p= get_var(var)))
25     return(FALSE);
    p->type= type;

    return(TRUE);
}

static int mark_term_var_type(P4TERM term, int type)
30 // mark the type of the given term (in the var-table) as given;
    // returns TRUE if the action done successfully; FALSE otherwise.
    {
        TabElem *p_tab_elem;

        if (p_tab_elem= get_term_tabelem(term))
35     {
            p_tab_elem->type= type;
            return(TRUE);
        }
    }

```

```

    }
else
    return(FALSE);
}

```

```

5 static int mark_term_list_type(List *var_lst, int type)
{    // mark the type of the variables in the given variable-term-list; return TRUE if ok,
  FALSE in error
  for( ; var_lst; var_lst= var_lst->next)
    mark_term_var_type((char *)var_lst->elem, type);

```

```

10 return(TRUE);
}

```

```

static List *cons(void *elem, List *lst)
{    // cons the given elem to the [front of the] list
  List *p;

```

```

15 if (!elem)
    return(lst);
if (p= calloc(1, sizeof(List)))
{
    p->elem= elem;
20 p->next= lst;
    p->elem_cnt= lst? lst->elem_cnt+1: 1;
}
else

```

```

25 {
    printf("\n***Unable to calloc List in cons()***\n");fflush(stdout);
    return(NULL);
}

```

```

return(p);
}

```

```

30 static List *ncons(List *lst, void *elem)
{    // ncons the given elem to the [end of the] list
  List *p, *l;

```

```

if (!elem)
    return(lst);
35 if (p= calloc(1, sizeof(List)))
{
    p->elem= elem;

```



```

    p->next= NULL;
    p->elem_cnt= 1;
    }
else
5   {
        printf("\n***Unable to calloc List in ncons()***\n");fflush(stdout);
        return(NULL);
    }
if (lst)
10  {
    lst->elem_cnt++;
    for(l= lst; l->next; l= l->next);
    l->next= p;
    }
15 else
    lst= p;

return(lst);
}

// --- action-routines ---

20 static int find_setof_soln(P4TERM goal)
{ // find (& buffer & reorder) the set of solns for the given goal;
  // return the Prolog-returned status of the call
  // Note that the P4 must have been initialized with the following query once before calling
  setof/3 or bagof/3:
25  //      def_array(tab_bagof, 100), record(block_limit, 0), record(last_bag_bound, 0).
  int i, soln_cnt, status;
  P4TERM var_set, R, vec;
  Value *p_vec;

  if (!doBufferSoln)
30  return(FALSE);

  for(var_set= check_term(p4make_nil()), i= varList_x-1; i>= 0; i--) // build a term for the set of
  var's
    var_set= check_term(p4make_dot((P4TERM) varList[i]->term, var_set));
  R= check_term(p4make_var());
35  goal= P4MAKE_FUNC_3("setof", var_set, goal, R);
  P4MAKE_CALL(goal);
  if ((status= (p4next_solution()== P4SESSION_SOLUTION)) && !p4is_nil(R))
  {      // found the solns - buffer them

```

```

for(soln_cnt= 0; !p4is_nil(R); R= p4cdr(R))
{
    if (p_vec= NEW_SOLN_VECTOR)
    { // enough room in the soln-buffer exists
5      for(vec= p4car(R), i= 0; !p4is_nil(vec); vec= p4cdr(vec), i++)
        {
            p_vec[i]= *get_term_value(p4car(vec));
        }
        soln_cnt++;
10    }
}
SET_BUF_SOLN_CNT(soln_cnt);
// **** how about ordering the solutions ----- in next pass
}

```

```

15 return(status);
}

```

```

static P4TERM split_var_range_term()
{ // return the split-var-range term (e.g. intsplit([X, Y]), realsplit([Z])) for the var's in the current
  constraint

```

```

20 // (useful only while solving with the interval solver.)
    // (we set the default var-type to be real.)
    // NOTE: if the variable is UNBOUND in the clause, a split (intsplit/realsplit)
    // may give rise to VB overflow !!

```

```

    int i;

```

```

25 P4TERM lst_term, term, intsplit, realsplit;
    //P4TERM anon;
    TabElem *p_tab_elem;

```

```

    if (!useIntervalSolver) // can we use intsplit/realsplit outside of the interval-solver
    ? NO
30    return(NULL);

```

```

    intsplit= realsplit= NULL;
    FOR_EACH_VAR(p_tab_elem, i)
        switch(p_tab_elem->type)

```

```

        {
35          case VAL_INTEGER:
            lst_term= check_term(p4make_dot(p_tab_elem->term, p4make_nil())); // X -> [X]
            term= P4MAKE_FUNC_3("intsplit", lst_term,
            check_term(p4make_atom_from_cstring("smallest_domain")),
            check_term(P4Make_Rational(p_tab_elem->precision)));
40            intsplit= P4AND(intsplit, term);
            break;

```

```

        case VAL_RATIONAL_FLOAT:
        case VAL_RATIONAL_FRACTION:
        case VAL_IRRATIONAL:
        case VAL_REAL:
5         lst_term= check_term(p4make_dot(p_tab_elem->term, p4make_nil())); // X -> [X]
            term= P4MAKE_FUNC_3("realsplit", lst_term,
check_term(p4make_atom_from_cstring("smallest_domain")),
check_term(P4Make_Rational(p_tab_elem->precision)));
            realsplit= P4AND(realsplit, term);
10         break;

        case VAL_LIST:
        case VAL_SYMBOL:
            break;

        default:
15         break;
    }
END_FOR_EACH_VAR(p_tab_elem, i)
    // <<--- how does it work in case the anon var. is nonnumber e.g. list ??
    // NOTE: if the variable is UNBOUND in the clause, a split (intsplit/realsplit) may give
20 rise to VB overflow !!
#ifdef NOIGNORE_ANON_SPLIT // don't do it until really needed
FOR_EACH_ANON_VAR(anon)
    lst_term= p4make_dot(anon, p4make_nil()); // X -> [X]
    term= P4MAKE_FUNC_3("realsplit", lst_term,
25 p4make_atom_from_cstring("smallest_domain"));
    realsplit= realsplit? P4AND(realsplit, term): term;
END_FOR_EACH_ANON_VAR(anon)
#endif /* NOIGNORE_ANON_SPLIT */

return(P4AND(intsplit, realsplit));
30 }

static P4TERM set_var_bounds_term()
{ // return the term setting bounds (e.g. -BOUND < X < BOUND) for all the variables used in the
constraint
// (useful only for the interval solver.)
35 // (ideally, we should first check to see if the var has a bound already set in the constraint.
// only when no such bound is set should we set a default one.
int i;
P4TERM term, bound;
//P4TERM anon;
40 TabElem *p_tab_elem;

```

```

if (!useIntervalSolver)
    return(NULL);

term= NULL;
FOR_EACH_VAR(p_tab_elem, i)
5  if (p_tab_elem->type== VAL_INTEGER || p_tab_elem->type== VAL_REAL ||
        p_tab_elem->type== VAL_RATIONAL_FLOAT || p_tab_elem->type==
VAL_RATIONAL_FRACTION)
    {
        bound= P4MAKE_FUNC_3("cc", p_tab_elem->term,
10  check_term(P4Make_Rational(DEF_LOWER_BOUND)),
        check_term(P4Make_Rational(DEF_UPPER_BOUND)));
        term= P4AND(term, bound);
    }
END_FOR_EACH_VAR(p_tab_elem, i)

15  // <<--- KNJ: try removing the constraints on anonymous var's    <<----
//FOR_EACH_ANON_VAR(anon)
// bound= P4MAKE_FUNC_3("cc", anon, P4Make_Rational(DEF_LOWER_BOUND),
P4Make_Rational(DEF_UPPER_BOUND));
// term= P4AND(term, bound);
20  //END_FOR_EACH_ANON_VAR(anon)
// <<--- KNJ: try removing the constraints on anonymous var's    <<----

return(term);
}

static P4TERM vars_multiple_of_precision_term()
25  {
    // return a term binding variables to be integer-multiples of their precision
    //      (e.g. int(_N), X= _N* X_precision)
    // NOTE: This constraint excludes irrational variables (e.g. PI)!

    int i;
    P4TERM term, t;
30  TabElem *p_tab_elem;

    term= NULL;
    FOR_EACH_VAR(p_tab_elem, i)
    if (p_tab_elem->ongrid && (p_tab_elem->type== VAL_REAL || p_tab_elem->type==
VAL_RATIONAL_FLOAT || p_tab_elem->type== VAL_RATIONAL_FRACTION))
35  {
        // for each variable, add: var_with_precision(Var, Precision).
        t= P4MAKE_FUNC_2("var_with_precision", p_tab_elem->term,
        check_term(P4Make_Rational(p_tab_elem->precision)));
        term= P4AND(term, t);
    }
40  END_FOR_EACH_VAR(p_tab_elem, i)

```

```

return(term);
}

```

```

static P4TERM buffered_func_terms()
{ // return the term containing the buffered functions (e.g. X= mean(X,Y) -> X= _R,
5 mean(_R,X,Y), where mean(_R,X,Y) is buffered)
return(combine_terms_array(funcTermBuf, funcTermBuf_x, TRUE, FALSE, NULL, FALSE));
}

```

```

static P4TERM get_exclude_solutions_term(int randomize)
{ // return a term corresponding to the negation of all the previous solutions of independent
10 var's e.g. (X/= 4, Y/=5, X/= 10, Y/=15).

```

```

TabElem *p_tab_elem;
P4TERM t, term;
Value *p_soln_vec;
int x_var, x_soln_vec;
15 int i;
char exclude_soln;

```

```

term= NULL;
exclude_soln= TRUE;
FOR_EACH_VAR(p_tab_elem, x_var)
20 if (p_tab_elem->is_independent_var)
{ // exclude only the variables to be used to generate uniq solns (e.q.
enumerate variables)

```

```

FOR_EACH_SOLN_VECTOR(p_soln_vec, x_soln_vec)
if (randomize) // randomize the exclusion of solutions
25 for(i=0, exclude_soln= FALSE; !exclude_soln && (i<
SolnDiffWt); i++)
{
exclude_soln= brandom();
if (exclude_soln && (t= get_value_term(&p_soln_vec[x_var])))
{
30 t= P4NEQ(p_tab_elem->term, t);
term= P4AND(term, t);
}
}

```

```

END_FOR_EACH_SOLN_VECTOR(p_soln_vec, x_soln_vec)
}
35 END_FOR_EACH_VAR(p_tab_elem, x_var)

```

```

return(term);
}

```

```

static P4TERM get_enum_ranges_term(int randomize)
{ // return a term corresponding to the enumerated-ranges (which were stored in the
40 enumRange buffer)

```

```

return(enumRangeTermBuf_CNT<=0? NULL: combine_terms_array(enumRangeTermBuf,
enumRangeTermBuf_CNT, TRUE, randomize, NULL, FALSE));
}

```

```

static P4TERM get_var_types_term(int randomize)
{
    // return a term corresponding to the combined var-types (e.g. int(X); neq_vars(X,Y))
    (which were stored in the VAR_TYPES buffer)

```

```

return(varTypesTermBuf_CNT<=0? NULL: combine_terms_array(varTypesTermBuf,
varTypesTermBuf_CNT, TRUE, randomize, NULL, FALSE));
}

```

```

static P4TERM pregoal_addenda_terms()
{
    // return terms (e.g. -BOUND< X < BOUND) to be added before the current goal
    P4TERM term, types_term, bound_term, prec_term;
    P4TERM exclude_solns_term, var_interval_term;

```

```

term= types_term= get_var_types_term(FALSE);
bound_term= NULL; // set_var_bounds_term(); <!-- KNJ: remove it for now <!--
term= P4AND(term, bound_term);
exclude_solns_term= RandomizeConstraints? get_exclude_solutions_term(TRUE): NULL;
term= P4AND(term, exclude_solns_term);
prec_term= vars_multiple_of_precision_term();
term= P4AND(term, prec_term);
var_interval_term= get_enum_ranges_term(RandomizeConstraints);
term= P4AND(term, var_interval_term);

return(term);
}

```

```

static P4TERM postgoal_addenda_terms()
{
    // return terms (e.g. intsplit/1, func-terms) to be added after the current goal
    P4TERM term, func_term;
    //P4TERM split_term;

```

```

term= func_term= buffered_func_terms();
// split_term= split_var_range_term(); // No need for splitting vars - and it may cause some
// overflow problems in VB
// term= P4AND(term, split_term);

return(term);
}

```

```

static P4TERM append_func_defs(P4TERM term)
{
    // append (before/after the term ?) the function defs to the given term

```

```

        // returns the appended term
P4TERM t, func_term;

func_term= buffered_func_terms();
t= P4AND(term, func_term);           // for now, append the func_term AFTER the given
5 term
INIT_funcTermBuf;           // function-terms already consumed here - reset the
functerms-buffer

return(t);
}

10 static P4TERM insert_func_defs(P4TERM var_defs, P4TERM expr_lst)
{ // insert the function def's between the var_defs and expr_lst
    // returns P4AND(append_func_defs(var_defs), expr_lst)
P4TERM term;

term= append_func_defs(var_defs);
15 term= P4AND(term, expr_lst);

return(term);
}

static BOOLEAN process_call_to_prolog(P4TERM goal)
{ // call Prolog IV with the given goal; return the result (i.e. TRUE/FALSE) of the call
20 // sets the BOOLEAN var resultFromProlog4 to the result.
P4TERM pregoal, postgoal;

resultFromProlog4= 0;
if (semError!= 0)
    return(FALSE);

25 if (!goal)
    return(FALSE);

if (pregoal= pregoal_addenda_terms()) // terms to add before goal
    goal = P4AND(pregoal, goal);
if (postgoal= postgoal_addenda_terms()) // terms to add after goal
30 goal = P4AND(goal, postgoal);

/*      -- P4 setof/3 & bagof/3 are buggy - hence we simulate them ourselves
if (doBufferSoln)
{      // call setof/3 - buffer (& order) all the solutions, return the result of call
return(resultFromProlog4= find_setof_soln(goal));
35 }

```

```

else
*/

```

```

P4MAKE_CALL(goal);
resultFromProlog4= p4next_solution();

```

```

5  /* -- Prints message incorrectly on time-limit-forced abortion
if (resultFromProlog4==P4SESSION_ERROR)
    {printf("\n***Encountered error in process_call_to_prolog(), errno: %d***\n",
p4errno);fflush(stdout);}
*/

```

```

10 return(resultFromProlog4== P4SESSION_SOLUTION);
}

```

```

static P4TERM combine_terms_array(P4TERM terms[], long size_terms, int do_conjunct, int
randomize, P4TERM var, int var_eq)
{
    // returns a combined terms for the elements (or, if var is non-NULL: var= Element (if var_eq is
    // TRUE) (or, if var_eq is FALSE, var != Element)) from the given array of terms :
    // use conjunct to combine terms if do_conjunct is TRUE, disjunct
    otherwise;
    // (randomize the array before combining if randomize is TRUE.)
    // (note that it shuffles the terms, when randomizing, in the given terms-buffer itself)

```

```

20 int i;
P4TERM t, cmbnd_term;
#define MAX_SHUFFLE (51)

```

```

if (size_terms<= 0)
    return(NULL);
25 if (size_terms==1)
    return(terms[0]);

```

```

if (randomize)
{
    int x, y, max_shuffle;
30 P4TERM tmp;
    // graduated scale for max-shuffling
    max_shuffle= (size_terms<= 10)? 3* size_terms:
    (size_terms<= 20)? 2* size_terms+

```

```

10:
35 MAX_SHUFFLE;
    for(i= 0; i< max_shuffle; i++)
    {
        // shuffle the terms in the terms-buffer
        x= random()% size_terms;
        y= random()% size_terms;
    }
}

```



```

        tmp= terms[x];
        terms[x]= terms[y];
        terms[y]= tmp;
    }
5    }
    for(cmbnd_term= NULL, i= 0; i< size_terms; i++)
    {
        t= !var? terms[i]: var_eq? P4EQ(var, terms[i]): P4NEQ(var, terms[i]);
        cmbnd_term= do_conjunct? P4AND(cmbnd_term, t): P4OR(cmbnd_term, t);
10    }

    return(cmbnd_term);
}

static P4TERM combine_terms_list(List *terms_lst, int do_conjunct, int randomize, P4TERM
var, int var_eq)
15 { // returns a combined terms for the elements (or, if var is non-NULL: var= Element (if var_eq is
TRUE) (or, if var_eq is FALSE, var != Element)) from the given list of terms :
    // use conjunct to combine terms if do_conjunct is TRUE, disjunct
    otherwise;
    // (randomize the array before combining if randomize is TRUE.)
20 P4TERM t, cmbnd_term;

    if (!randomize)
    {
        for(cmbnd_term= NULL; terms_lst; terms_lst= terms_lst->next)
        {
25            t= !var? terms_lst->elem: var_eq? P4EQ(var, terms_lst->elem): P4NEQ(var,
terms_lst->elem);
            cmbnd_term= do_conjunct? P4AND(cmbnd_term, t): P4OR(cmbnd_term, t);
        }
    }
30 else
    { // put the list-terms in an array, call combine_terms_array() with the array
      P4TERM *terms;
      List *lst;
      int i;

35      for(i= 0, lst= terms_lst; lst; lst= lst->next, i++);
      if (!(terms= calloc(i, sizeof(P4TERM))))
      {
          printf("\n***Unable to calloc P4TERM-array in
combine_terms_list()***\n");fflush(stdout);
40          return(NULL); // return NULL on error
      }
    }

```

```

        for(i= 0, lst= terms_lst; lst; lst= lst->next, i++)
            terms[i]= lst->elem;
        cmbnd_term= combine_terms_array(terms, i, do_conjunct, randomize, var, var_eq);
        free(terms);
5      }
    return(cmbnd_term);
}

static char *map_func_name(char *given_func_name)
    // check if the given function-name needs to be mapped e.g. gcd -> gcdtemp
10    // return the properly mapped function name
{
    int i;

    for(i= 0; FuncRenameList[i].func_name && strcmp(given_func_name,
    FuncRenameList[i].func_name); i++);
15    return(FuncRenameList[i].func_name? FuncRenameList[i].map_to_name: given_func_name);
}

static P4TERM unop_term(int op, P4TERM term)
{ // make unary op term for standard op's, return the result term (e.g. "- X" => "uminus(Res,
X)", return Res).
20    // (Note: This function can handle only the PrologIV-standard unary operators; use
make_funcor() for nonstandard operators (e.g. abs, factorial) )
    char *func;
    P4TERM res;

    if (!term)
25    {
        semError= ERR_NULL_TERM;
        return(NULL);
    }

    switch(op)
30    { // at some point, we need to decide between approx. & exact solvers (e.g. uminuslin vs
    uminus)
        case '-': func= useIntervalSolver? "-.": "-"; break;
        case '+': func= useIntervalSolver? "+.": "+"; break;
        default: return(NULL);
35    }
    func= map_func_name(func);
    if (!(res= P4MAKE_FUNC_1(func, term)))
    {
40        semError= ERR_MAKING_FUNCUTOR;
        return(NULL);
    }

```

```

    }
    return(res);
}

static P4TERM binop_arith_term(P4TERM term1, int op, P4TERM term2)
5 { // make binary op term for standard arithmetic op's, return the result term (e.g. " X + Y" =>
  "plus(Res, X, Y)".
    // Returns (ptr to the term representing) the result of the computation rather than the
    Boolean status of the call.
    // Note that these are called using arity 3 Prolog IV predicates.
10    // (Note: This function can handle only the PrologIV-standard binary operators; use
    make_funcor() for nonstandard operators (e.g. mod, div) )
    char *func;
    P4TERM res;

    if (!term1 || !term2)
15    {
        semError= ERR_NULL_TERM;
        return(NULL);
    }

    switch(op)
20    { // at some point, we need to decide between approx. & exact solvers (e.g. minuslin vs minus)
        case '+': func= useIntervalSolver? "+.": "+"; break;
        case '-': func= useIntervalSolver? "-.": "-"; break;
        case '*': func= useIntervalSolver? ".*": "*"; break;
        case '/': func= useIntervalSolver? " ./ ": "/"; break;
25    case '%': func= "mod"; break;
        case '\\': func= "intdiv"; break;
        case '^': func= ".^."; break; // power/2
        // case '~': func= "~"; break;
        default: return(NULL);
30    }
    func= map_func_name(func);
    if (!(res= P4MAKE_FUNC_2(func, term1, term2)))
    {
        semError= ERR_MAKING_FUNCTOR;
35    return(NULL);
    }
    return(res);
}

static P4TERM binop_bool_term(P4TERM term1, int op, P4TERM term2)
40 { // make binary op term for standard boolean op's (e.g. " X, Y" => "and(X, Y)".
    char *func;

```

P4TERM res;

if (!term1 || !term2)

```
{
  semError= ERR_NULL_TERM;
  return(NULL);
}
```

switch(op)

```
{ // at some point, we need to decide between approx. & exact solvers (e.g. minuslin vs minus)
  case EQ: func= "="; break;
  case NEQ: func= "dif"; break;
  case ',': func= ","; break; // note that ,/2 is different from and/2 in Prolog IV
  case ';': func= ";"; break; // note that ;/2 is different from or/2 in Prolog IV
  case LT: func= useIntervalSolver? "lt": "ltlin"; break;
  case LE: func= useIntervalSolver? "le": "lelin"; break;
  case GT: func= useIntervalSolver? "gt": "gtlin"; break;
  case GE: func= useIntervalSolver? "ge": "gelin"; break;
  default: return(NULL);
}
```

func= map_func_name(func);

if (!(res= P4MAKE_FUNC_2(func, term1, term2)))

```
{
  semError= ERR_MAKING_FUNCTOR;
  return(NULL);
}
```

return(res);

}

static P4TERM make_termlist2term(List *lst)

```
{ // convert the given list of P4-terms to Prolog IV list-term
  // recursive implementation to maintain the order of elem's
  // free the given list ??
```

```
return(lst? check_term(p4make_dot((P4TERM) lst->elem, make_termlist2term(lst->next))):
check_term(p4make_nil()));
}
```

static P4TERM make_valslst2term(List *lst)

```
{ // convert the given list of Values to Prolog IV list-term
  // recursive implementation to maintain the order of elem's
return(lst? check_term(p4make_dot(get_value_term(lst->elem), make_valslst2term(lst->next))):
check_term(p4make_nil()));
}
```

static P4TERM make_int_rel(List *lst)

```

{ // make (& return) a conjunct of int/1 terms (e.g. int(X), int(Y)) for the given list
P4TERM t, term;

for(term= NULL; lst; lst= lst->next)
{
5      t= P4MAKE_FUNC_1("int", (P4TERM) lst->elem);
      term= P4AND(term, t);
}

// add the actual int_rel-term to the VAR_TYPES buffer - so we can later manipulate it
if (term)
10      ADD_VAR_TYPES_TERM(term);

//      return just a placeholder since we already added the int_rel-term to the buffer
return(P4TRUE);
}

static P4TERM make_real_rel(List *lst)
15 { // make (& return) a conjunct of real/1 terms (e.g. real(X), real(Y)) for the given list
P4TERM t, term;

for(term= NULL; lst; lst= lst->next)
{
20      t= P4MAKE_FUNC_1("real", (P4TERM) lst->elem);
      term= P4AND(term, t);
}

// add the actual real_rel-term to the VAR_TYPES buffer - so we can later manipulate it
if (term)
      ADD_VAR_TYPES_TERM(term);

25 //      return just a placeholder since we already added the real_rel-term to the buffer
return(P4TRUE);
}

static P4TERM make_rational_rel(List *lst)
30 { // make (& return) a conjunct of real/1 & rational/1 terms (e.g. real(X), rational(X)) for the
given list
P4TERM t, term;

for(term= NULL; lst; lst= lst->next)
{
35      t= P4MAKE_FUNC_1("rational", (P4TERM) lst->elem);
      term= P4AND(term, t);
}

```

```
// add the actual real_rel-term to the VAR_TYPES buffer - so we can later manipulate it
if (term)
```

```
    ADD_VAR_TYPES_TERM(term);
```

```
//      return just a placeholder since we already added the real_rel-term to the buffer
```

```
return(P4TRUE);
```

```
}
```

```
static P4TERM make_neqvars_rel(List *varslst)
```

```
{      // make a term to enforce all var's in the list are different from each other;
```

```
      //      put that term in VARS_TYPE_BUF for later use; return P4TRUE for now at end
```

```
P4TERM t, term;
```

```
List *list1, *list2;
```

```
term= NULL;
```

```
for(list1= varslst; list1; list1= list1->next)
```

```
    for(list2= list1->next; list2; list2= list2->next)
```

```
        {
            if (list1->elem && list2->elem)
```

```
                {
                    t= P4NEQ((P4TERM)list1->elem, (P4TERM)list2->elem);
```

```
                    term= P4AND(term, t);
```

```
                }
```

```
        }
```

```
// add the actual neqvars-term to the VAR_TYPES buffer - so we can later manipulate it
```

```
if (term)
```

```
    ADD_VAR_TYPES_TERM(term);
```

```
//      return just a placeholder since we already added the neqvars-term to the buffer
```

```
return(P4TRUE);
```

```
}
```

```
static P4TERM make_eqvars_rel(List *varslst)
```

```
{      // make a term to enforce all var's in the list are equal to each other;
```

```
      //      put that term in VARS_TYPE_BUF for later use; return P4TRUE for now at end
```

```
P4TERM t, term;
```

```
List *list1, *list2;
```

```
term= NULL;
```

```
for(list1= varslst; list1; list1= list1->next)
```

```
    for(list2= list1->next; list2; list2= list2->next)
```

```
        {
            if (list1->elem && list2->elem)
```

```
                {
```

```

        t= P4EQ((P4TERM)list1->elem, (P4TERM)list2->elem);
        term= P4AND(term, t);
    }
}

5 // add the actual eqvars-term to the VAR_TYPES buffer - so we can later manipulate it
  if (term)
      ADD_VAR_TYPES_TERM(term);

  // return just a placeholder since we already added the eqvars-term to the buffer
  return(P4TRUE);
10 }

static P4TERM make_neqvarvals_rel(P4TERM var, List *valslist)
{
    // make a term to specify that the given var is not equal to any of the given values in the
    // valslist;
    // put that term in VARS_TYPE_BUF for later use; return P4TRUE for now at end
15 P4TERM t, term;
    List *list1;

    if (!var || !valslist)
        return(P4TRUE);

    term= NULL;
20 for(list1= valslist; list1; list1= list1->next)
    {
        t= P4NEQ(var, (P4TERM)list1->elem);
        term= P4AND(term, t);
    }

25 // add the actual eqvars-term to the VAR_TYPES buffer - so we can later manipulate it
  if (term)
      ADD_VAR_TYPES_TERM(term);

  // return just a placeholder since we already added the eqvars-term to the buffer
  return(P4TRUE);
30 }

static P4TERM make_optimizable_rel(P4TERM rel)
{
    // optimize the given rel-term (for now, we just put it at the start of the clauses);
    // put that term in VARS_TYPE_BUF for later use; return P4TRUE for now at end
  if (!rel)
35 return(P4TRUE);

  // add the actual term to the VAR_TYPES buffer - so we can later manipulate it

```

```
ADD_VAR_TYPES_TERM(rel);
```

```
//      return just a placeholder since we already added the term to the buffer
return(P4TRUE);
}
```

```
5 static P4TERM make_inlist(P4TERM var, List *lst)
{ // make (& return) (var inlist lst) constraint; (lst is list of P4 terms.);
  // (Obsolete: the translation uses inlist/1 e.g. "X in [a, b, 5]" => "X ~ inlist([a, b, 5])".
  //      that approach provoked many problems in relations which expected
  constants e.g. gcd/2.)
10  // Obsolete: return(P4MAKE_FUNC_2("inlist", var, make_termlist2term(lst)));
  // The current translation essentially uses disjunction e.g. "X in [a, b, 5]." => (X= a; X= b;
  X= 5)."
```

```
return(P4MAKE_FUNC_2("one_of", var, make_termlist2term(lst)));
}
```

```
15 static P4TERM make_fromlist(P4TERM var, List *lst)
{ // make (& return) (var inlist lst) constraint; (lst is list of P4 terms.);
  // some modifications/optimizations are made for randomization
  // mark the given var as an independent variable
  TabElem *p_tab_elem;
20  P4TERM term;

  if (!(p_tab_elem= get_term_tabelem(var)))
    return(NULL);
  p_tab_elem->is_independent_var = TRUE;      // all enumerated var's are considered
  independent
```

```
25 term= P4MAKE_FUNC_2("random", var, make_termlist2term(lst));

  // add the actual term to the enumRangeTerm buffer -
  //      so we can later randomize it (to help produce different-looking solutions)
  ADD_ENUM_RANGE_TERM(term);
```

```
//      return just a placeholder since we already added the enumRange term to the buffer
30 return(P4TRUE);

}
```

```
static P4TERM make_notinlist(P4TERM var, List *lst)
{ // make (& return) (var notinlist lst) constraint; (lst is list of P4 terms.);
  // ?? should we try the conjunction e.g. "X notin [a, b, 5]." => (X/= a, X/= b. X/= 5)."
```

```
35 ???
```



```

return(P4MAKE_FUNC_2("outlist", var, make_termlist2term(lst)));
}

static P4TERM make_interval(P4TERM left, int left_rel, P4TERM var, int right_rel, P4TERM
right)
5 {      // make (& return) interval constraint e.g. "4.5 < X <= 9" => "oc(X, 4.5, 9)"; "4.6 >= X >
2" => "oc(X, 2, 4.6)"
      // Note that this is different from an enumeration over an interval.
char pred_name[3];
P4TERM term;

10 if (!var || ((left_rel==LE || left_rel==LT) && (right_rel==GE || right_rel==GT)) ||
      ((left_rel==GE || left_rel==GT) && (right_rel==LE || right_rel==LT)))
    {      // e.g. "4.5 < X > 7.5"
      semError= ERR_INVALID_INTERVAL;
      return(NULL);
15    }

      // build the predicate name from the given relationships
pred_name[0]= (left_rel==LE || left_rel==GE)? 'c':'o';
pred_name[1]= (right_rel==LE || right_rel==GE)? 'c':'o';
pred_name[2]= 0;

20 term= P4MAKE_FUNC_3(pred_name, var, left, right);

#ifdef WANT_ALL_SMALL_SOLUTIONS
      // while this ok to do (and gives good solutions), it leads to unnecessary
      undesirably small solutions in large quantity
      // --- For now, don't use it ---
25 if (useIntervalSolver)
    {      // split the bounded var when using interval-solver
      char *split_pred;
      P4TERM t, lst_term;
      TabElem *p_tab_elem;

30      if (!(p_tab_elem= get_term_tabelem(var)))
          return(NULL);
      split_pred= (p_tab_elem->type== VAL_INTEGER)? "intsplitted": "realsplitted";
      lst_term= p4make_dot(var, p4make_nil()); // X -> [X]
      t= P4MAKE_FUNC_3(split_pred, lst_term,
35      p4make_atom_from_cstring("smallest_domain"), P4Make_Rational(p_tab_elem->precision));
      term= P4AND(term, t);
    }
#endif /* WANT_ALL_SMALL_SOLUTIONS */

return(term);

```

```

}

static P4TERM make_enumeration(P4TERM left, int left_rel, P4TERM var, int right_rel,
P4TERM right, P4TERM step)
{
    // make (& return) enumeration constraint e.g. "[4.5 < X <= 8 step 1]" => "(X=4.5;
5 X=5.5; X= 6.5; X= 7.5)"
    // Note that this is different from interval.
    char *pred_name;
    TabElem *p_tab_elem;
    P4TERM term, adj_left, adj_right;

10 if (!var || !step || ((left_rel==LE || left_rel==LT) && (right_rel==GE || right_rel==GT)) ||
    ((left_rel==GE || left_rel==GT) && (right_rel==LE || right_rel==LT)))
    {
        // e.g. "4.5 < X > 7.5"
        semError= ERR_INVALID_INTERVAL;
        return(NULL);
15 }

if (!(p_tab_elem= get_term_tabelem(var)))
    return(NULL);
p_tab_elem->is_independent_var = TRUE;           // all enumerated var's are considered
independent
20 // Note: we don't care here which is min or max value - enumerate/4 takes care of that.
pred_name= enumerateVarsRandomlyNoHistory? "enumerate_random": "enumerate";
if (p_tab_elem->type== VAL_INTEGER)
    pred_name= enumerateVarsRandomlyNoHistory?
    "enumerate_int_random": "enumerate_int";

25 adj_left= (left_rel==LT)? P4MAKE_FUNC_2("+", left,
check_term(P4Make_Rational(p_tab_elem->precision))):
    (left_rel==GT)? P4MAKE_FUNC_2("-", left,
check_term(P4Make_Rational(p_tab_elem->precision))):
    left;
30 adj_right= (right_rel==LT)? P4MAKE_FUNC_2("+", right,
check_term(P4Make_Rational(p_tab_elem->precision))):
    (right_rel==GT)? P4MAKE_FUNC_2("-", right,
check_term(P4Make_Rational(p_tab_elem->precision))):
    right;

35 term= P4MAKE_FUNC_4(pred_name, var, adj_left, adj_right, step);
// add the actual enumerated-range term to the enumRangeTerm buffer -
// so we can later randomize it (to help produce different-looking solutions)
ADD_ENUM_RANGE_TERM(term);

// return just a placeholder since we already added the enumRange term to the buffer

```

```

return(P4TRUE);
}

```

```

static P4TERM make_exteval(P4TERM left, int left_rel, P4TERM var, int right_rel, P4TERM
right)

```

```

5 { // make (& return) exteval constraint e.g. not("4.5 < X <= 9") => "outoc(X, 4.5, 9)"; not("4.6
>= X > 2)" => "outoc(X, 2, 4.6)"
char pred_name[6];

```

```

if (!var || ((left_rel==LE || left_rel==LT) && (right_rel==GE || right_rel==GT)) ||
((left_rel==GE || left_rel==GT) && (right_rel==LE || right_rel==LT)))
10 { // e.g. "4.5 < X > 7.5"
semError= ERR_INVALID_INTERVAL;
return(NULL);
}

```

```

strcpy(pred_name, "out");
15 pred_name[3]= (left_rel==LE || left_rel==GE)? 'c':'o';
pred_name[4]= (right_rel==LE || right_rel==GE)? 'c':'o';
pred_name[5]= 0;

return(P4MAKE_FUNC_3(pred_name, var, left, right));
}

```

```

20 static P4TERM make_func(char *pred_name, List *arg_lst)
{ // make & return PrologIV functor for the given predicate/args (e.g. given "mean", [X, Y]
for "mean(X, Y)")
// We allow no user-relations - only user-functions.
// As such, we add 1 result var (at the leftmost position) to all the user-functions.
25 // free the given list after use ??

```

```

int i, cnt, arity;
P4TERM terms[MAX_ARITY+1], anon_result, func_term;
List *lst;

```

```

anon_result= NULL;
30 cnt= 0;
arity= arg_lst? arg_lst->elem_cnt: 0;
if (arity > MAX_ARITY)
{
semError= ERR_ARITY_TOO_MANY;
35 return(NULL);
}

```

```

// convert given func/n to rel/n+1 e.g. X= mean(X, Y) -> X= _R, mean(_R, X, Y).
terms[cnt++]= anon_result= check_term(p4make_var());

```

```

for(i= 0, lst= arg_lst; i< MAX_ARITY && lst; lst= lst->next,i++)
    terms[cnt++]= (P4TERM)lst->elem;

pred_name= map_func_name(pred_name);
func_term= check_term(p4vmake_functor(cnt, p4str2symbol(pred_name), terms));
5  ADD_funcTermBuf(func_term);    // add func_term to funcTermBuf so we can generate code
    at end for it
    return(anon_result);
}

static P4TERM indexed_list_element(P4TERM list, List *index_lst)
10 {    // return PrologIV term for the given list-indexed-term (e.g. L[1, 2])
    // Translation scheme: List[I, J, K] => nth(K, nth(J, nth(I, List))).
    P4TERM nth_elem, func_term;

    if (!list || !index_lst)
        return(NULL);

    nth_elem= check_term(p4make_var());
    func_term= P4MAKE_FUNC_3("nth", nth_elem, index_lst->elem, list);
    ADD_funcTermBuf(func_term);    // add func_term to funcTermBuf so we can generate code
    at end for it

    return(index_lst->next? indexed_list_element(nth_elem, index_lst->next): nth_elem);
20 }

static P4TERM if_then_else(P4TERM cond, P4TERM then_term, P4TERM else_term)
{ // make & return PrologIV term for the given if-then-else construct
  // This is a backtrack-less-implementation of if-then-else.
  //    if C then T else E => (C, T, !); E.

25 return(else_term? P4IF_THEN_ELSE(cond, then_term, else_term): P4IF_THEN(cond,
    then_term));
}

static P4TERM if_then_elseif(P4TERM then_cond, P4TERM then_term, P4TERM else_cond,
P4TERM else_term)
30 { // make & return PrologIV term for the guarded-if if-then-elseif construct
  //    if TC then T elseif EC E => (TC, T); (EC, E).

    return(P4IF_THEN_ELSEIF(then_cond, then_term, else_cond, else_term));
}

```

%}

%union

{

int ival;

float fval;

double dval;

char *string;

List *list;

P4TERM term;

}

%token <ival> INTNUM NOTIN_SET IN_SET FROM_SET NOT PI

%token <ival> INT_PRED REAL_PRED RATIONAL_PRED

FRACTION_PRED LIST_PRED FREEZE

%token <ival> IF THEN ELSE ELSEIF SUCCEED FAIL SYMBOL_PRED

END_VAR_DEFS STEP

%token <ival> EQVARS_PRED NEQVARS_PRED NEQVARVALS_PRED

OPTIMIZABLEREL_PRED

%token <ival> ONGRID_PRED OFFGRID_PRED

%token <string> REALNUM ATOM_CONST VAR

%token <ival> GT LT

%token <ival> GE ">="

%token <ival> LE "<="

%token <ival> EQ "=="

%token <ival> NEQ "!="

%token <ival> EXRANGE_START "[!"

%left ':'

%left ','

%nonassoc '~'

%nonassoc '=='

%left "==" '<' '>' ">=" "<="

%nonassoc '|'

%left '+ ' -'

%left '*'

%left '/'

%left '%' '\\'

%nonassoc NEG

%left '^'

%left '!'

%type <term> prolog_expr expr expr_lst rel_expr range_expr type_list type_symbol

%type <term> rel_expr arith_expr function p_list type_int type_real type_rational type_fraction

%type <term> constant num_constant atom var if_then_else flow_expr type_expr base_expr

```

%type <term> index type_eqvars type_neqvars type_neqvarvals type_optimizablerel
%type <term> type_ongrid type_offgrid
%type <ival> rel_op_lege
%type <dval> number
5 %type <list> list var_lst constant_lst

%%
prolog_expr :      expr_lst '!'                                {$$=
(P4TERM)process_call_to_prolog($1);}
10      |      expr_lst ',' END_VAR_DEFS ',' expr_lst '!'
      {$$= (P4TERM)process_call_to_prolog(insert_func_defs($1, $5));}
      |      error
      {$$= NULL; }
      ;

15  expr_lst:      expr_lst ',' expr_lst      {$$= P4AND($1, $3);}
      |      expr_lst ',' expr_lst      {$$= P4OR($1, $3);}
      |      expr
      ;

20  expr  : '(' expr_lst ')'                                {$$= $2;}
      |      base_expr
      {$$= append_func_defs($1);}
      ;

25  base_expr:  rel_expr
      |      range_expr
      |      flow_expr
      |      type_expr
      ;

30  flow_expr :      if_then_else
      |      FREEZE                                {$$=
P4CUT;}
      |      SUCCEED                                {$$=
P4TRUE;}
      |      FAIL                                    {$$=
P4FAIL;}
35      ;

type_expr :      type_int
      |      type_real
      |      type_rational
      |      type_fraction
40      |      type_list

```

```

5      type_symbol
      type_ongrid
      type_offgrid
      type_eqvars
      type_neqvars
      type_neqvarvals
      type_optimizablerel
      ;

10  if_then_else: IF expr THEN expr ELSE expr      {$$= if_then_else($2, $4, $6);}
      |          IF expr THEN expr ELSEIF expr THEN expr      {$$=
if_then_elseif($2, $4, $6, $8);}
      |          IF expr THEN expr
      |          {$$= if_then_else($2, $4, NULL);}
      ;

15  type_int      :      INT_PRED var_lst ')'      {mark_term_list_type($2,
VAL_INTEGER); $$= make_int_rel($2);}
      ;
      type_real      :      REAL_PRED var_lst ')'      {mark_term_list_type($2,
VAL_RATIONAL_FLOAT); $$= make_real_rel($2);}
20  ;
      type_rational :      RATIONAL_PRED var_lst ')'      {mark_term_list_type($2,
VAL_RATIONAL_FLOAT); $$= make_rational_rel($2);}
      ;
      type_fraction: FRACTION_PRED var_lst ')'      {mark_term_list_type($2,
25  VAL_RATIONAL_FRACTION); $$= make_real_rel($2);}
      ;
      type_list      :      LIST_PRED var_lst ')'      {$$= P4TRUE;
mark_term_list_type($2, VAL_LIST);}
      ;
30  type_symbol :      SYMBOL_PRED var_lst ')'      {$$= P4TRUE;
mark_term_list_type($2, VAL_SYMBOL);}
      ;
      type_ongrid: ONGRID_PRED var_lst ')'      {$$= P4TRUE;
35  set_term_on_grid($2);}
      ;
      type_offgrid: OFFGRID_PRED var_lst ')'      {$$= P4TRUE;
reset_term_on_grid($2);}
      ;

40  type_eqvars :      EQVARS_PRED var_lst ')'      {$$= make_eqvars_rel($2);}
      ;
      type_neqvars: NEQVARS_PRED var_lst ')'      {$$= make_neqvars_rel($2);}
      ;

```



```

        {$$= LT;}
        |
        |      "<="
        |      {$$= LE;}
        |      '>'
5         {$$= GT;}
        |
        |      ">="
        |      {$$= GE;}
        ;

p_list: '[' list ']'                                {$$=
10 make_termlist2term($2);}
    ;

list    :    list ',' arith_expr                    {$$= ncons($1, (void *)$3);}
        |    arith_expr
        |    {$$= list((void *)$1);}
15    ;

arith_expr    :    arith_expr '+' arith_expr        {$$= binop_arith_term($1, '+', $3);}
        |    arith_expr '-' arith_expr            {$$=
        binop_arith_term($1, '-', $3);}
        |    arith_expr '/' arith_expr            {$$=
20    binop_arith_term($1, '/', $3);}
        |    arith_expr '*' arith_expr            {$$=
        binop_arith_term($1, '*', $3);}
        |    arith_expr '^' arith_expr            {$$=
25    binop_arith_term($1, '^', $3);}
        |    arith_expr '%' arith_expr            {$$=
        make_functor("mod", cons((void *)$1, list((void *)$3)));}
        |    arith_expr '\\' arith_expr           {$$=
30    make_functor("intdiv", cons((void *)$1, list((void *)$3)));}
        |    '-' arith_expr    %prec NEG          {$$= unop_term('-',
        $2);}
        |    arith_expr '!'
        |    {$$= make_functor("factorial", list((void *)$1));}
        |    '|' arith_expr '|'                   {$$=
35    make_functor("abs", list((void *)$2));}
        |    '(' arith_expr ')'
        |    {$$= $2;}
        |    function
        |    {$$= $1;}
        |    index
        |    {$$= $1;}
40    |    p_list
        |    {$$= $1;}

```

```

        |      constant
        |      { $$ = $1; }
        |      var
        |      { $$ = $1; }
5      ;

function      :      ATOM_CONST '(' list ')'      { $$ = make_function($1, $3); }
        |      ATOM_CONST '(' ' ' )
        { $$ = make_function($1, NULL); }
        ;

10     index      :      var '[' list ']'      { $$ =
indexed_list_element($1, $3); }
        ;

var_lst      :      var_lst ',' var      { $$ =
ncons($1, (void *)$3); }
15     |      var
        { $$ = list((void *)$1); }
        ;

constant_lst:  constant_lst ',' constant      { $$ = ncons($1, (void *)$3); }
        |      constant
20     { $$ = list((void *)$1); }
        ;

constant      :      num_constant
        |      atom
        ;

25     num_constant:      INTNUM      { Value val; $$ =
check_term(p4make_int($1)); val.type= VAL_INTEGER; val.value.integer= $1;
insert_const($$, &val); }
        |      REALNUM      { Value val;
30     $$ = check_term(p4make_rational($1)); /* yylex passes string in yylval */ val.type=
VAL_RATIONAL_FLOAT; val.value.rational.real= atof($1); insert_const($$, &val); }
        |      PI      { Value
val; $$ = check_term(P4Make_Rational(PI_VAL)); val.type= VAL_RATIONAL_FLOAT;
val.value.rational.real= PI_VAL; insert_const($$, &val); }
        ;

35     atom      :      ATOM_CONST      { $$ =
check_term(p4make_atom_from_cstring($1)); }
        ;

```

```

var          :      VAR
                                     {$$= insert_var($1);}
          |      '{ VAR ',' number }'
insert_var_with_precision($2, $4);}
5          |      '{ VAR ',' number '/' number }'      {$$=
insert_var_with_precision($2, ((double)$4)/$6);}
          ;

number:      INTNUM                                     {$$= (double)$1;}
          |      REALNUM                                 {$$=
10 (double)atof($1); /* yylex passes string in yylval */}
          |      PI                                     {$$=
(double)PI_VAL;}
          ;

%%
15 static int restart_parser()
{
  yyclearin;
  return(1);
}

20 static int restart_lexer()
{
  extern void yyrestart(FILE *);

  yyrestart(NULL);
  return(1);
25 }

static BOOLEAN init_solve_constraint(int keep_solns)
{ // initialize all the buffers; if keep_solns is true, do not initialize the solns (& value) buffers
  // returns true if ok, false in error
  if (!StartProlog4Session(NULL))
30     return(FALSE);
  // initialize various data structures
  if (!keep_solns)
  {
35     INIT_valBuf;
    INIT_solnBuf;
  }
  INIT_inExprBuf;
  INIT_vars;
  INIT_funcTermBuf;

```

```

INIT_anonVarBuf;
INIT_constBuf;
INIT_enumRangeTermBuf;
INIT_varTypesTermBuf;

```

```

5    // initialize parser/lexer states & data structures (if any) ...
restart_lexer();
restart_parser();

return(TRUE);
}

```

```

10  int yyerror(char *s)
    { // handle error
      // printf("syntax error\n");
      semError= ERR_PARSE;
      return(0);
15  }

```

```

        // some augmented functions for Prolog IV API - some of them may go away as we get
        newer, better API
static P4SYMBOL p4str2symbol(char *str)    // pseudo p4-routine
{ // convert the given atomic (i.e. starting with lowercase e.g. 'aTom') string to Prolog IV symbol
20  P4SYMBOL sym;
    if (p4cstring_to_symbol(str, &sym))
    {
        semError= ERR_GETTING_TERM;
        return(0);
25  }
    return(sym);
}

```

```

static P4TERM p4make_atom_from_cstring(char *str)    // pseudo p4-routine
{ // convert the given atomic (i.e. starting with lowercase e.g. 'aTom') C-string to Prolog IV
30  atomic term
    P4SYMBOL sym;
    if (p4cstring_to_symbol(str, &sym))
    {
        semError= ERR_GETTING_TERM;
35  return(NULL);
    }
    return(p4make_atom(sym));
}

```

```

static int p4is_constant(P4TERM term)    // pseudo p4-routine

```

```

{ // returns TRUE if the given term is constant
int type;

type= P4WHAT_IS(term);
return(!(type==P4NULL || type==P4NOTATERM || type== P4VAR || type==
5 P4UNEXPECTEDTERM));
}

static P4TERM P4Make_Rational(double val) // pseudo p4-routine
{ // KNJ: make (& return) term for the given double (rational) value
char buf[256];

10 sprintf(buf, "%.6f", val); // maximum precision is limited to 6 positions
return(make_rational_strfloat(buf));
}

static P4TERM p4make_rational(char *floatstring)
{ // by Pascal Bouvier (of Prologianet): Build a positive rational number from a C-string
15 containing its decimal
// representation (e.g. "4.5", "1.0000000000", "24.02e1997", ...)
char buf[256];

strcpy(buf, floatstring);
return(make_rational_strfloat(buf));
20 }

static double p4val_as_double(P4TERM T)
{ // by Pascal Bouvier (of Prologianet): Given a numeric Term, converts it as a double (with
probable loss of precision)
switch (P4WHAT_IS(T))
25 {
case P4INTEGER: case P4RATIONAL:
return(nearest_ip_fpd(dereference(T)));

default:
return(-11111.111); /* in error */

30 }
}

static char *p4_symbol_to_cstring(P4SYMBOL symbol)
{ // by Pascal Bouvier (of Prologianet): return the c-string rep. of the symbol
return(symbol_shortidP(symbol));
35 }
}

```

```

' hlp4API.h
/*
 * hlp4API.h: Specification of high level API to Prolog IV
 *
5  */

#include <windows.h>                // necessary for VB stuff e.g. BSTR

// The stdcall calling-convention is used for compatibility with VB
#define CCONV_stdcall

#define DEF_PRECISION                (0.01)
10 #define DEF_SOLN_DIFF_WT          (1)
#define DEF_FLOAT_INTERVAL_STEP      (0.1)
#define DEF_INTEGER_INTERVAL_STEP    (1)
#define DEF_UPPER_BOUND              (64000)
#define DEF_LOWER_BOUND              (-64000)
15 #define DEF_GRID                   (TRUE)

typedef struct s_list
{ // ordered list
  short elem_cnt;    // total no. of elements (including this element) in the list
  void *elem;        // this element (type to be inferred from the context)
20  struct s_list *next;
} List;

typedef struct s_functor
{ // structure to represent a Prolog IV functor (e.g. member/2) in C
  char *predicate;
25  int arity;
} Functor;

typedef struct s_rational
{ // structure to represent a rational number
  double real;    // real representation of a rational e.g. "4.5"
30  long num;      // numerator from A/B rep. of rational e.g. 9 from 9/2
  long den;      // denominator from A/B rep. of rational e.g. 2 from 9/2
} Rational;

typedef struct s_bound    // a bound for non-rational real
{
35  char is_infinite;    // flag - true if the real-val is infinite
  double val;          // value of the non-infinite real
} Bound;

```

```

typedef struct s_real          // representation (bound) for non-rational real
{
    Bound lower, upper;
} Real;

5  typedef struct s_val
    {
        int type;                // type of the result (e.g. VAL_INTEGER, VAL_LIST)
        union {
            long integer;        // e.g. 5
10         Rational rational;    // e.g. 9/2 = 4.5
            Real real;           // e.g. (lower: 2.5, upper: 5)
            char *string;        // atom e.g. area
            Functor functor;     // e.g. add(X, Y)
            List *list;          // value itself is a list of values e.g. [a, [x, y], 5]
15         } value;
    } Value;

        // Types of value
#define VAL_UNKNOWN            0
#define VAL_INTEGER            10
20 #define VAL_RATIONAL_FLOAT    12
#define VAL_RATIONAL_FRACTION  13
#define VAL_IRRATIONAL         14
#define VAL_REAL                15        // not used in the Value structure
#define VAL_STRING              20
25 #define VAL_LIST              25
#define VAL_FUNCTOR             30
#define VAL_SYMBOL              35
#define VAL_VAR                 100
#define DEF_VAR_TYPE    VAL_UNKNOWN    // default type for untyped variables

30 char * CCONV GetHLAPIVersion(); // return the current version of the Prolog HL API
    BSTR CCONV VBGetHLAPIVersion(); // VB wrapper for GetHLAPIVersion()

    // StartProlog4Session: starts Prolog IV, return true (1) if ok, false (0) otherwise.
        // p4hlapilib_file is the pathname to the high-level Prolog IV API library file
        // (MUST call one of: {StartProlog4Session, StartProlog4SessionSetStacks} before
35 starting Prolog IV.)
int CCONV StartProlog4Session(char *p4hlapilib_file);
    // StartProlog4SessionSetStacks: starts Prolog IV, return true (1) if ok, false (0) otherwise.
        // p4hlapilib_file is the pathname to the high-level Prolog IV API library file
        // heapsize is the heap-stack size; choicesize is the choice-stack size.
40        // (MUST call one of: {StartProlog4Session, StartProlog4SessionSetStacks} before
starting Prolog IV.)

```

```
int CCONV StartProlog4SessionSetStacks(char *p4hlapilib_file, long heapsize, long choicesize);
```

```
    // Error return-values from SolveConstraint() (keep them all negative)
```

```
#define ERR_INITIALIZATION -10
```

```
#define ERR_CONSTRAINT_TOO_LONG -15
```

```
#define ERR_GETTING_TERM -20
```

```
#define ERR_MAKING_FUNCTOR -25
```

```
#define ERR_INVALID_INTERVAL -30
```

```
#define ERR_ARITY_TOO_MANY -35
```

```
#define ERR_PARSE -40
```

```
#define ERR_NULL_TERM -45
```

```
// SolveConstraint: solve the given constraint (e.g. "X= Y+ 4, Y=2.") using a linear/interval solver as needed;
```

```
    // backtrack over the previous solution if the given constraint is NULL.
```

```
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
```

```
    // returns negative integer in error (e.g. if the constraint could not be parsed).
```

```
int CCONV SolveConstraint(char *constraint);
```

```
// SolveConstraintLin: solve the given constraint (e.g. "X= Y+ 4, Y=2.") using a linear solver only;
```

```
    // backtrack over the previous solution if the given constraint is NULL.
```

```
    // return true (=1) [false (=0)] if the constraint is [un]solvable;
```

```
    // returns negative integer in error (e.g. if the constraint could not be parsed).
```

```
int CCONV SolveConstraintLin(char *constraint);
```

```
// SolveConstraintOrdered: solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as needed
```

```
    // present the solutions conforming to the given order (e.g. ORDER_DIFF_TOGETHER)
```

```
    // backtrack over the previous solution if the given constraint is NULL.
```

```
    // returns, on first call (i.e. when constraint is non-NULL), (1+ the_total_count_of_solutions) (> 0) [false (=0)] if the constraint is [un]solvable;
```

```
    // (note that in case of constraints without variables (e.g. "4= 4."), total-no.-of-solutions is 0, though the constraint is provable.)
```

```
    // returns, on subsequent calls (i.e. when constraint is NULL), true (= 1) [false (=0)] if a solution exists [does not exist];
```

```
    // returns negative integer in error (e.g. if the constraint could not be parsed).
```

```
int CCONV SolveConstraintOrdered(char *constraint, int order_type);
```

```
// SolveConstraintOrderedNSolns: solve the given constraint (e.g. "X= Y+ 4, Y=2."); using a linear/interval solver as needed
```

```
    // solve to find the given maximum no. (= max_soln) of solutions
```

```
    // backtrack over the previous solution if the given constraint is NULL.
```

```
    // present the solutions conforming to the given order (e.g. ORDER_DIFF_TOGETHER)
```

```
    // returns, on first call (i.e. when constraint is non-NULL), (1+
```



```

the_total_count_of_solutions) (> 0) [false (=0)] if the constraint is [un]solvable;
//      (note that in case of constraints without variables (e.g. "4= 4."),
total-no.-of-solutions is 0, though the constraint is provable.)
// returns, on subsequent calls (i.e. when constraint is NULL), true (= 1) [false (=0)] if a
5  solution exists [does not exist];
//      returns negative integer in error (e.g. if the constraint could not be parsed).
int CCONV SolveConstraintOrderedNSolns(char *constraint, int order_type, int max_soln);

```

```

// order-types
#define NO_ORDER 0
10 #define ORDER_DIFF_TOGETHER 10
#define ORDER_LIKE_TOGETHER 20
#define ORDER_RANDOM 30
#define ORDER_UNIQ_SOLUTIONS 40

```

```

// set the precision for solving the constraint & for the solutions in the real domain
15 // returns TRUE if ok, FALSE otherwise
int CCONV SetPrecision(double precision);

```

```

// set the weight to indicate how "different" the solutions must be from each other in
Uniq_Soln_Order
//      (the higher the weight, the more the solutions are "different".)
20 // returns TRUE if ok
int CCONV SetSolnDiffWt(int soln_diff_wt);

```

```

// fractionalize the rationals if do_fractionalize is TRUE; not otherwise (fractionalization may
slow things down a bit.)
int CCONV FractionalizeRational(int do_fractionalize);

```

```

25 // IsFullyConstrained: returns TRUE if the given constraint is fully constrained (i.e.
solvable & all variables are constant); FALSE otherwise (or in error)
int CCONV IsFullyConstrained(char *constraint);

```

```

// return TRUE (1) if the given variable is independent (i.e. specified in an enumeration);
FALSE (0) otherwise
30 int CCONV IsIndependentVar(char *var);

```

```

// GetValue: return the ptr to the value (in Value structure) of the given variable (e.g. "Area") if
known;
Value * CCONV GetValue(char *var);

```

```

// return type (e.g. VAL_INTEGER) of the given Value;
35 //      returns VAL_UNKNOWN in error
long CCONV GetValue_type(Value *val);

```

```

        // return type (e.g. VAL_INTEGER) of the given variable;
        //      returns VAL_UNKNOWN in error
long CCONV GetVarValue_type(char *var);

```

```

5 // GetValue_int: return integer value of the given Value structure
  // return ERR_GETVALUE_INT in error (e.g. given structure is not integer)
long CCONV GetValue_int(Value *value);
#define ERR_GETVALUE_INT    (-111111765)

```

```

10 // GetValue_rational: return rational value of the given Value structure
  // return <ERR_GETVALUE_RAT, ERR_GETVALUE_INT, ERR_GETVALUE_INT>
  in error (e.g. given structure is not rational)
Rational CCONV GetValue_rational(Value *value);
#define ERR_GETVALUE_RAT    (-111111765.9876)

```

```

15 // return float rep. of the given rational Value
double CCONV GetValue_rational_float(Value *val);

```

```

  // return numerator of the fractional rep. of the given rational Value
long CCONV GetValue_rational_numer(Value *val);

```

```

  // return denominator of the fractional rep. of the given rational Value
long CCONV GetValue_rational_denom(Value *val);

```

```

20 // return real value (i.e. lower & upper bound) from the given non-rational Value
  structure;

```

```

        //      return <1, 0> in error (e.g. given structure is not real)
Real CCONV GetValue_real(Value *val);

```

```

#define ERR_GETVALUE_REAL    (-111111765.9876)

```

```

25 // return lower bound for the given non-rational real value
  // return ERR_GETVALUE_REAL in error or when the lower bound is infinite
double CCONV GetValue_real_lower(Value *val);

```

```

  // return upper bound for the given non-rational real value
  // return ERR_GETVALUE_REAL in error or when the upper bound is infinite
30 double CCONV GetValue_real_upper(Value *val);

```

```

// GetValue_string: return (uniform) string representation of the given Value structure
  // return NULL in error (e.g. given structure is not valid)
char * CCONV GetValue_string(Value *value);
BSTR CCONV VBGetValue_string(Value *value); // VB wrapper for GetValue_string()

```

```

35 // GetVarValue: return (uniform) string representation of the given variable

```

// return NULL in error

char * CCONV GetVarValue(char *var);

BSTR CCONV VBGetVarValue(char *var); // VB wrapper for GetVarValue()

// return (uniform) string representation of the given variable in the value-buffer

5 int CCONV GetVarValueBuf(char *var, int valuebuf_len, char valuebuf[]); // return length
(>0) of value-string if ok; <= 0 in error

// PrintAllVarVals: Print all the var's with their values in the given buffer; return ptr to the given
buffer

// (assumes the buffer is large enough to store all the var-value pairs.)

10 char * CCONV PrintAllVarVals(char buf[]);

BSTR CCONV VBPrintAllVarVals(); // VB wrapper (almost) for PrintAllVarVals()

// PrintAllVarVals: Print all the var's with their values in an allocated buffer; return ptr to the
given buffer

// (assumes the buffer is large enough to store all the var-value pairs.)

15 char * CCONV PrintAllVarValsAllocate();

// Compile: compile & load the given p4_filename (containing Prolog IV program)

// return true (1) if the file compiled ok, false (0) in error

int CCONV Compile(char *p4_filename);